gnuplot 5.4

An Interactive Plotting Program

Thomas Williams & Colin Kelley

Version 5.4 organized by: Ethan A Merritt and many others

Major contributors (alphabetic order):
Christoph Bersch, Hans-Bernhard Bröker,
John Campbell, Robert Cunningham,
David Denholm, Gershon Elber,
Roger Fearick, Carsten Grammes,
Lucas Hart, Lars Hecking, Péter Juhász,
Thomas Koenig, David Kotz,
Ed Kubaitis, Russell Lang, Timothée Lecomte,
Alexander Lehmann, Jérôme Lodewyck,
Alexander Mai, Bastian Märkisch,
Ethan A Merritt, Petr Mikulík,
Daniel Sebald, Carsten Steger, Shigeharu Takeno,
Tom Tkacik, Jos Van der Woude,
James R. Van Zandt, Alex Woo, Johannes Zellner

Mailing list for comments: `gnuplot-info@lists.sourceforge.net`
Web site and issue trackers: `http://sourceforge.net/projects/gnuplot`

This manual was originally prepared by Dick Crawford.

Version 5.4 (May 2021)

Contents

Part I

Gnuplot

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 and 5.0 additions:
           See list of contributors at head of this document.
```

Introduction

Gnuplot is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms. The source code is copyrighted but freely distributed (i.e., you don't have to pay for it). It was originally created to allow scientists and students to visualize mathematical functions and data interactively, but has grown to support many non-interactive uses such as web scripting. It is also used as a plotting engine by third-party applications like Octave. Gnuplot has been supported and under active development since 1986.

Gnuplot supports many types of plots in either 2D and 3D. It can draw using lines, points, boxes, contours, vector fields, surfaces, and various associated text. It also supports various specialized plot types.

Gnuplot supports many different types of output: interactive screen terminals (with mouse and hotkey input), direct output to pen plotters or modern printers, and output to many file formats (eps, emf, fig, jpeg, LaTeX, pdf, png, postscript, ...). Gnuplot is easily extensible to include new output modes. Recent additions include interactive terminals based on wxWidgets (usable on multiple platforms), and Qt. Mouseable plots embedded in web pages can be generated using the svg or HTML5 canvas terminal drivers.

The command language of **gnuplot** is case sensitive, i.e. commands and function names written in lowercase are not the same as those written in capitals. All command names may be abbreviated as long as the abbreviation is not ambiguous. Any number of commands may appear on a line, separated by semicolons (;). Strings may be set off by either single or double quotes, although there are some subtle differences. See **syntax (p. 59)** and **quotes (p. 59)** for more details. Example:

```
set title "My First Plot";  plot 'data';  print "all done!"
```

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 (p. 32)**). 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 **help** 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 built-in help on any topic, type **help** followed by the name of the topic or **help ?** to get a menu of available topics.

A large set of demo plots is available on the web page `http://www.gnuplot.info/demo/`

When run from command line, gnuplot is invoked using the syntax

```
gnuplot {OPTIONS} file1 file2 ...
```

where file1, file2, etc. are input file as in the **load** command. On X11-based systems, you can use

```
gnuplot {X11OPTIONS} {OPTIONS} file1 file2 ...
```

see your X11 documentation and **x11 (p. 296)** in this document.

Options interpreted by gnuplot may come anywhere on the line. Files are executed in the order specified, as are commands supplied by the -e option, for example

```
gnuplot   file1.in   -e "reset"   file2.in
```

The special filename "-" is used to force reading from stdin. **Gnuplot** exits after the last file is processed. If no load files are named, **Gnuplot** takes interactive input from stdin. See help **batch/interactive (p. 30)** for more details. The options specific to gnuplot can be listed by typing

```
gnuplot --help
```

See **command-line-options (p. 30)** for more details.

In sessions with an interactive plot window you can hit 'h' anywhere on the plot for help about **hotkeys** and **mousing** features. Section **seeking-assistance** will help you to find further information, help and FAQ.

Seeking-assistance

The canonical gnuplot home page can be found at `http://www.gnuplot.info`

Before seeking help, please check file FAQ.pdf or the above website for a `FAQ (Frequently Asked Questions) list.`

Another resource for help with specific plotting problems (not bugs) is

```
https://stackoverflow.com/questions/tagged/gnuplot
```

Bug reports and feature requests should be uploaded to the trackers at

```
http://sourceforge.net/projects/gnuplot/support
```

Please check previous reports to see if the bug you want to report has already been fixed in a newer version.

When reporting a bug or posting a question, please include full details of the gnuplot version, the terminal type, and the operating system. A short self-contained script demonstrating the problem is very helpful.

Instructions for subscribing to gnuplot mailing lists may be found via the gnuplot development website on SourceForge `http://sourceforge.net/projects/gnuplot`

Please note that before you write to any of the gnuplot mailing lists you must first subscribe to the list. This helps reduce the amount of spam.

The address for mailing to list members is:

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

A mailing list for those interested in the development version of gnuplot is:

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

New features

Features introduced in version 5.4

These sections list new commands, plot styles, and other features introduced in version 5.4.

Support for 64-bit integer arithmetic

- All evaluation of expressions and functions uses 64-integer arithmetic if supported by the platform.
- Integer overflow is detected and handled according to user preference. See **overflow (p. 179)**.

Voxel grids

Gnuplot now supports operations based on 3D grids of voxel data.

- **set vgrid $gridname size N** creates an NxNxN grid of voxels.
- **set vxrange** [**vxmin:vxmax**] together with **set vyrange** and **set vzrange** define which region of space the grid occupies. This may or may not be identical to the xyz range of the plot.
- **voxel(x,y,z)** can be used in expressions to read or write an individual voxel.
- **vfill DATA_SOURCE using x:y:z:radius:(<expression>)** acts analogously to a plot command except that instead of plotting it increments voxels near each point in the input data.
- **vclear $gridname** resets an existing voxel grid to contain all zero values.
- the current contents of one or more voxel grids can be referenced by **splot** commands to assign colors or other properties of plot elements by using the **voxel** function in using specifiers. See demo **voxel.dem**.
- voxel grids can also be plotted by name in **splot** commands with plot styles **dots**, **points**, or **isosurface**. See demo **vplot.dem**.

New plot styles and style options

- 3D plot style **with polygons** reads polygon faces from a data file. This can be used to create a surface or to construct a solid object. See **with polygons (p. 78)**.

- **splot $voxelgrid with {dots|points}** marks all voxels whose value is above a requested threshold level.
- **splot $voxelgrid with isosurface** creates a tessellated 3D surface enclosing voxels above a requested threshold level. See **isosurface (p. 84)**.
- Voxel grid values can be referenced in the **using** specifiers for 3D plots.
- **set spiderplot** selects a new plotting mode allowing creation of spider plots (also known as radar charts). These are essentially parallel axis plots where the axes are arranged radially rather than vertically. See **spiderplot (p. 79)**, **set style spiderplot (p. 203)**, **set paxis (p. 186)**.
- Plot style **with circles** can be used in 3D plots.
- Plot style **with boxes** can be used in 3D plots.
- 2D plot style **with arrows** is identical to **with vectors** except that each arrow is specified using x:y:length:angle rather than x:y:xdelta:ydelta
- splot FOO with pm3d fillcolor <colorspec>
- pm3d surfaces can have individual fillstyle and separate top/bottom fillcolor
- pm3d option **noclipcb** causes quadrangles with palette color outside cbrange to be skipped rather than being drawn with color clipped to cbmin or cbmax.
- Customized contour line types. See **set cntrparam (p. 137)**.

New data pre-processing filters

- **smooth zsort** sorts 2D points on values in a 3rd column. See **smooth zsort (p. 113)**.

New commands and command options

- Voxel grid commands. See **set vgrid (p. 210)**, **set vxrange (p. 212)**, **vclear (p. 234)**, **vfill (p. 235)**, and **voxel (p. 39)**.
- New options for showing the xy xz and yz planes in 3D plots. See **set walls (p. 212)**, **set grid vertical (p. 153)**.
- **set table separator {tab|comma|"char"}** can be used to create csv files. See **plot with table (p. 204)**.
- New options **set view projection {xy|xz|yz}** adjust view angles, axis tic and label placement to generate a 2D projection of a 3D splot. **set view projection xy** is equivalent to **set view map**.
- **set rgbmax <value>** controls interpretation of input RGB values.
- Array size can be implicit if an initializer is present, e.g. **Array A = [1,2,3]**.
- Optional radial clipping of line segments in polar mode. See **set clip (p. 135)**.
- Extra lines to customize the key can be added by substituting **keyentry** in place of a filename or function in **plot** and **splot** commands. This produces a line in the key without generating a corresponding plot. See **keyentry (p. 159)**.
- User-specified translation of mouse coordinates (EXPERIMENTAL). See map_projection demo.
- **set datafile columnheaders** causes first line of input to be read as strings rather than as data values. Equivalent to **set key autotitle columnheader** except that it does not affect generation of key entries. If this option is in effect the **stats** command will generate an array of strings containing the column headers found.
- You can define multiple textbox styles. See **set style textbox (p. 203)**.

New terminals and terminal options

- The **pcl5** terminal has been extended to support PCL5e/PCL5c printers and many modern gnuplot features.
- The **pstricks** terminal has been extended to support many modern gnuplot features including RGB colors and transparency, filled polygons, and boxes.
- New terminal **pict2e** to use the LaTeX2e pict2e environment. It directly supersedes older terminals **latex**, **emtex**, **eepic**, and **tpic**, which are no longer built by default.
- The **texdraw** terminal has been extended to support text at arbitrary angles, variable line width, v5 dashed lines, and filled boxes and polygons. It can now be used with plain TeX, too.
- The previously experimental Direct2D variant of the **windows** terminal replaces the GDI and GDI+ variants. It now supports printing using D2D and color fonts.
- The **pm** OS/2 terminal has been modernized to support e.g. uft8, bold and italic text, and dashed lines. (Since 5.2.7).
- The **dospc** and **svga** DOS terminals have been modernized and now support interactive keyboard and mouse (svga only) input.

Pixmaps as objects

- **set pixmap** allows import of an image in standard format (png jpeg gif) as a pixmap that can be positioned anywhere in a plot or on the page. Unlike plotting **with image**, pixmap objects retain their original aspect ratio and size independent of axis scaling or rotation. See **pixmap (p. 186)**.

Week-date time formats

The Covid-19 pandemic of 2020/2021 generated increased interest in plotting epidemiological data, which is often tabulated using a "week date" reporting convention. This revealed deficiencies with gnuplot support for this convention, including errors in time formats %W and %U. These formats worked incorrectly prior to version 5.4.2.

- Time specifier format %W has been brought into accord with the ISO 8601 week date standard.
- Time specifier format %U has been brought into accord with the CDC/MMWR week date standard.
- New function **tm_week(time, std)** returns ISO or CDC standard week of year.
- New function **weekdate_iso(year, week, day)** converts ISO standard week date to calendar time.
- New function **weekdate_cdc(year, week, day)** converts CDC standard week date to calendar time.

Other new features

- Enhanced text mode accepts \U+xxxx (xxxx is a 4 or 5 character hexadecimal) as representing a Unicode code point that is converted to the corresponding UTF-8 byte sequence on output.
- The character sequence $# in a **using** specifier evaluates to the total number of columns available in the current line of data. For example "plot FOO using 0:(column($# - 1))" plots the last-but-one field of each row.
- Time/date input recognizes format %p to handle am/pm field
- plot titles are evaluated *after* plotting, rather than before. This allows the title to reference quantities calculated while plotting.
- Built-in modified Bessel functions (besi0 besi1 besin)
- Built-in order N Bessel functions of the 1st and 2nd kind (besjn besyn)

Changes

- **pm3d filled area quadrangles** are clipped smoothly to current zrange. This affects pm3d surfaces and also the faces of 3D boxes, polygons, etc.
- Revised syntax for plot style 'with parallelaxes'. See **parallel (p. 76)**. The histogram, parallelaxis, and spiderplot styles now use similar syntax that can iterate over plot elements: **plot for [column=1:N] DATA using column**
- Sampling generated by pseudofile '+' is affected by **set trange**.
- Offsets from **set offsets** are applied only to autoscaled axes. The documentation has always said this, but it was not applied consistently.
- Imaginary values returned by the using specifier of a 2D plot are treated as undefined values (NaN) rather than as real(value). This was always true for function plots and 3D data plots. E.g. the following two plots are equivalent. plot [-1:1] sqrt(x); plot [-1:1] '+' using 1:(sqrt($1)
- The **set fontpath** command is deprecated. The search path for fonts to be embedded in output from the postscript terminal has been revised.

Features introduced in version 5.2

New plot styles and style options

- 3D plot style **with zerrorfill**. See **zerrorfill (p. 85)**, **fenceplots (p. 84)** and `zerror demo.`
- Beeswarm plots. See **set jitter (p. 156)**, **beeswarm (p. 61)** and `beeswarm plot demo`
- The symbol used for individual points in a plot can be controlled by data values (see **pointtype variable (p. 78)**)

New data pre-processing filters

- Normalized frequency of occurrence in a data set (see **smooth fnormal (p. 112)**)
- Automated binning of data (see **bins (p. 107)**)

Polar mode improvements and extensions

- Polar coordinates may be used in label, arrow, and object definitions
- **set [m]ttics** places ticmarks and labels on the perimeter of a polar plot. See `polar axis and ticlabels demo`
- **set rlabel (p. 194)** places a label above the r axis
- Inverted **rrange (p. 194)** (i.e. set rrange [90:0]) allows use of celestial horizontal coordinates. See `solar path demo`
- **set border polar (p. 133)** draws a solid line around the perimeter of a polar plot
- **set theta (p. 206)** controls the position of theta = 0 around the perimeter of a polar plot and the sense (clockwise or anti-clockwise) of increasing theta

Nonlinear coordinates systems

- Any plot axis can be assigned a pair of functions, possibly nonlinear, that describe the forward and reverse mapping to a linear range (see **set nonlinear (p. 174)**) `Nonlinear x/y axis demo`
- The familiar command **set logscale** has been reimplemented as a special case nonlinear axis where the paired functions are log(x) and exp(x).

New commands and command options

- Inside the bracketed clause of an iteration, **continue** jumps immediately to the next iteration, **break** immediately exits from the iteration
- **toggle {<plotno> | "plottitle" | all}"** interactively enables or disables display of one element of the current plot (see **toggle (p. 233)**)
- **save fit** replaces deprecated command **update**
- **set table "outfile.name" append** will append subsequent tabulated plots to an existing text file rather replacing its contents
- **set pm3d lighting** describes a lighting model with specular highlighting (see **lighting (p. 189)**)
- **set minussign** tells gnuplot to use a special symbol in the current encoding to replace the ascii character '-' in negative numbers
- **set micro** tells gnuplot to use a special symbol in the current encoding to replace the ascii character 'u' for the scientific notation prefix "micro" The special typographic symbols for micro and minussign are used only in axis tic labels and strings explicitly created with gprintf(). The byte sequence used to represent these characters depends on the current encoding.

New data type "array"

- This gnuplot version introduces a new data type **array name**[**size**]. An array must be declared before use. Each array element A[i] may be a string, an integer, a real number, or a complex value. A single array may contain elements with different types. The cardinality operator |A| returns the size of array A. See **arrays (p. 44)**.

New terminals and terminal options

- See **sixelgd (p. 284)** for description of a new terminal that supports interleaving plots with the command lines that generated them if gnuplot is run inside a vt340-compatible terminal emulator
- The **domterm (p. 249)** terminal supports interleaving plots with the command lines that generated them if gnuplot is run inside an svg-aware terminal emulator
- The **windows (p. 290)** terminal supports saving the current graph to a bitmap file
- The **windows (p. 290)** terminal graph window can be docked to the wgnuplot text window
- New (experimental) Direct2D/DirectWrite backend for the **windows** terminal
- The **wxt** terminal supports exporting to an EMF file or printer on Windows
- The **dumb** terminal supports ANSI colors for lines and fill area
- The **tkcanvas** terminal has been rewritten to support many more modern gnuplot features, as well as new languages. (Since 5.0.3)

Other new features

- An additional rotation angle **azimuth** affects the orientation of 3D plots. This can be set from the command line (see **set view azimuth (p. 211)**) or by dragging with the right mouse button.
- gnuplot running under Windows can interpret Unicode (BMP) input scripts by converting them to the current encoding from **set encoding**, including UTF-8
- Textboxes can be assigned a border color and fill color (see **set style textbox (p. 203)**)
- Customized plot legends (see **plot title (p. 123)**, **set key (p. 157)**, **multiple keys (p. 161)**)
- A sampling range specifier for plotting with pseudofile '+' can include a sampling interval. For example: plot sample [t=0:100:10] '+' using (t):(1):(label[t]) with labels

- Pseudo-file '++' generates samples on the u and v axes, rather than x and y. This allows placement of multiple parametric surfaces in 3D that occupy distinct regions of Cartesian space. See **sampling.dem**.
- new formats descriptors tH tM tS handle relative times (interval lengths). See **time_specifiers (p. 151)**.
- ^R initiates a reverse-search through the history for the built-in readline which is used on Windows, too, see **command-line-editing (p. 32)**.
- Revised printing support on Windows using **set output "PRN"**, see **windows printing (p. 292)**.

Features introduced in version 5.0

- The dot-dash pattern of a line can now be specified independent of other line properties. See **dashtype (p. 50)**, **set dashtype (p. 140)**, **set linetype (p. 164)**
- The default sequence of colors used for successive elements in a plot is more easily distinguished by users with color-vision defects. The color sequence is under user control (see **set colorsequence (p. 135)**). This mechanism can also be used to generate monochrome plots (see **set monochrome (p. 168)**). In previous gnuplot versions **monochrome** could only be selected when changing the current terminal via **set terminal**.
- New plot styles **with parallelaxes**, **with table**, and labeled contours.
- New data pre-processing filter for monotonic cubic splines (see **smooth mcsplines (p. 112)**)
- Text markup now supports bold and italic font settings in addition to subscript, superscript, font size and other previously available properties. Enhanced text mode is now enabled by default. See **enhanced text (p. 34)**. Text elements can be enclosed in a box (see **set style textbox (p. 203)**).
- Interactive terminals support hypertext labels that only appear when the mouse hovers over the label's anchor point.
- New coordinate system (Degrees, Minutes, Seconds). See **set xtics geographic (p. 220)**.
- The default format for axis labels is "% h" ("$%h$" for LaTeX terminals). This format is like the C standard format %g except that the exponential term, if present, is written using a superscript. E.g. 1.2 x 10^5 rather than 1.2E05.
- Command scripts may place in-line data in a named data block for repeated plotting. See **inline data (p. 47)**.
- Support for 32-bit Alpha channel + RGB color #AARRGGBB. See **colorspec (p. 49)**.
- Support for HSV color space via a translation function hsv2rgb(H,S,V).
- Secondary axes (x2, y2) may be locked to the primary axis via a mapping function. In the simplest case this guarantees that the primary and secondary axis ranges are identical. In the general case it allows you to define a non-linear axis, something that previously was possible only for log scaling. See **set link (p. 165)**.
- Each function in a plot command may optionally be preceded by a sampling range. This does not affect the overall range of the plot, only the range over which this function is sampled. See **plot (p. 101)** and **piecewise.dem**.
- If the external library libcerf is available, it is used to provide complex math routines cerf, cdawson, erfi, faddeeva, and the Voigt profile VP(x,sigma,gamma).
- The **import** command attaches a user-defined function name to a function provided by an external shared object (support is operating-system dependent). A template header and example source and make files for creating a suitable external shared object are provided in the demo collection.
- Previous commands in the history list of an interactive session can be reexecuted by number. For example, **history !5** will reexecute the command numbered 5 in the **history** list.
- Bit-shift operators >> and <<.

- Shell invocation of gnuplot can pass parameters to a gnuplot script. gnuplot -c scriptfile.gp ARG1 ARG2 ARG3 ...

Differences between versions 4 and 5

Some changes introduced in version 5 may cause certain scripts written for earlier versions of gnuplot to behave differently.

* Revised handling of input data containing NaN, inconsistent number of data columns, or other unexpected content. See Note under **missing (p. 141)** for examples and figures.

* Time coordinates are stored internally as the number of seconds relative to the standard unix epoch 1-Jan-1970. Earlier versions of gnuplot used a different epoch internally (1-Jan-2000). This change resolves inconsistencies introduced whenever time in seconds was generated externally. The epoch convention used by a particular gnuplot installation can be determined using the command **print strftime("%F",0)**. Time is now stored to at least millisecond precision.

* The function **timecolumn(N,"timeformat")** now has 2 parameters. Because the new second parameter is not associated with any particular data axis, this allows using the **timecolumn** function to read time data for reasons other than specifying the x or y coordinate. This functionality replaces the command sequence **set xdata time; set timefmt "timeformat"**. It allows combining time data read from multiple files with different formats within a single plot.

* The **reverse** keyword of the **set [axis]range** command affects only autoscaling. It does not invert or otherwise alter the meaning of a command such as **set xrange [0:1]**. If you want to reverse the direction of the x axis in such a case, say instead **set xrange [1:0]**.

* The **call** command is provides a set of variables ARGC, ARG0, ..., ARG9. ARG0 holds the name of the script file being executed. ARG1 to ARG9 are string variables and thus may either be referenced directly or expanded as macros, e.g. @ARG1. The contents of ARG0 ... ARG9 may alternatively be accessed as array elements ARGV[0] ... ARGV[ARGC]. An older gnuplot convention of referencing call parameters as tokens $0 ... $9 is deprecated.

* The optional bandwidth for the kernel density smoothing option is taken from a keyword rather than a data column. See **smooth kdensity (p. 113)**.

Deprecated syntax

Gnuplot version 4 deprecated certain syntax used in earlier versions but provided a configuration option that allowed backward compatibility. Support for the old syntax has now been removed.

Deprecated in version 4 and removed in version 5:

```
set title "Old" 0,-1
set data linespoints
plot 'file' thru f(x)
plot 1 2 4               # horizontal line at y=1
update
```

Current equivalent:

```
TITLE = "New"
set title TITLE offset char 0, char -1
set style data linespoints
plot 'file' using 1:(f(column(2)))
plot 1 linetype 2 pointtype 4
save fit "filename"
```

Deprecated in version 5

```
if (defined(VARNAME)) ...
set style increment user
call 'script' 1.23 ABC
   (in script:  print $0, "$1", "number of args = $#")
set fontpath
set clabel
fit control variables FIT_*
```

Current equivalent:

```
if (exists("VARNAME")) ...
set linetype
call 'script' 1.23 "ABC"
   (in script:  print ARG1, ARG2, "number of args = ", ARGC
set cntrlabel
set fit <option> <value>
```

Deprecated in version 5.4

```
# use of a file containing `reread` to perform iteration
N = 0;  load "file-containing-reread";
file content:
    N = N+1
    plot func(N,x)
    pause -1
    if (N<5) reread
```

Current equivalent

```
do for [N=1:5] {
    plot func(N, x)
    pause -1
}
```

Demos and Online Examples

The **gnuplot** distribution contains a collection of examples in the **demo** directory. You can browse on-line versions of these examples produced by the png, svg, and canvas terminals at `http://gnuplot.info/demos`

The commands that produced each demo plot are shown next to the plot, and the corresponding gnuplot script can be downloaded to serve as a model for generating similar plots.

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 either program options (see command-line-options) or names of files containing **gnuplot** commands. Each file or command string will be executed in the order specified. The special filename "-" is indicates that commands are to be read from stdin. **Gnuplot** exits after the last file is processed. If no load files and no command strings are specified, **gnuplot** accepts interactive input from stdin.

Command line options

Gnuplot accepts the following options on the command line

```
-V, --version
-h, --help
-p  --persist
-d  --default-settings
-s  --slow
-e  "command1; command2; ..."
-c  scriptfile ARG1 ARG2 ...
```

-p tells the program not to close any remaining interactive plot windows when the program exits.

-d tells the program not to execute any private or system initialization (see **initialization (p. 55)**).

-s tells the program to wait for slow font initialization on startup. Otherwise it prints an error and continues with bad font metrics.

-e "command" tells gnuplot to execute that single command before continuing.

-c is equivalent to -e "call scriptfile ARG1 ARG2 ...". See **call (p. 86)**.

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
```

To give **gnuplot** commands directly in the command line, using the "-persist" option so that the plot remains on the screen afterwards:

```
gnuplot -persist -e "set title 'Sine curve'; plot sin(x)"
```

To set user-defined variables a and s prior to executing commands from a file:

```
gnuplot -e "a=2; s='file.png'" input.gpl
```

Canvas size

This documentation uses the term "canvas" to mean the full drawing area available for positioning the plot and associated elements like labels, titles, key, etc. NB: For information about the HTML5 canvas terminal see **set term canvas (p. 242)**.

In earlier versions of gnuplot, some terminal types used the values from **set size** to control also the size of the output canvas; others did not. The use of 'set size' for this purpose was deprecated in version 4. Almost all terminals now behave as follows:

set term <terminal_type> size <XX>, <YY> controls the size of the output file, or "canvas". By default, the plot will fill this canvas.

set size <XX>, <YY> scales the plot itself relative to the size of the canvas. Scale values less than 1 will cause the plot to not fill the entire canvas. Scale values larger than 1 will cause only a portion of the plot to fit on the canvas. Please be aware that setting scale values larger than 1 may cause problems.

Example:

```
set size 0.5, 0.5
set term png size 600, 400
set output "figure.png"
plot "data" with lines
```

These commands produce an output file "figure.png" that is 600 pixels wide and 400 pixels tall. The plot will fill the lower left quarter of this canvas. This is consistent with the way multiplot mode has always worked.

Command-line-editing

Command-line editing and command history are supported using either an external gnu readline library, an external BSD libedit library, or a built-in equivalent. This choice is a configuration option at the time gnuplot is built.

The editing commands of the built-in version are given below. Please note that the action of the DEL key is system-dependent. The gnu readline and BSD libedit libraries have their own documentation.

Command-line Editing Commands	
Character	Function
	Line Editing
`^B`	move back a single character.
`^F`	move forward a single character.
`^A`	move to the beginning of the line.
`^E`	move to the end of the line.
`^H`	delete the previous character.
`DEL`	delete the current character.
`^D`	delete current character. EOF if line is empty.
`^K`	delete from current position to the end of line.
`^L`	redraw line in case it gets trashed.
`^U`	delete the entire line.
`^W`	delete previous word.
`^V`	inhibits the interpretation of the following key as editing command.
`TAB`	performs filename-completion.
	History
`^P`	move back through history.
`^N`	move forward through history.
`^R`	starts a backward-search.

Comments

The comment character # may appear almost anywhere in a command line, and **gnuplot** will ignore the rest of that line. A # does not have this effect inside a quoted string. Note that if a commented line ends in '\' then the subsequent line is also treated as part of the comment.

See also **set datafile commentschars (p. 143)** for specifying a comment character for data files.

Coordinates

The commands **set arrow**, **set key**, **set label** and **set object** 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**, **polar**, **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 x2,y2 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 **set xyplane (p. 221)**); **screen** specifies the screen area (the entire area — not just the portion selected by **set size**), with 0,0 at bottom left and 1,1 at top right. **character** coordinates are used primarily for offsets, not absolute positions. The **character** vertical and horizontal size depend on the current font.

polar causes the first two values to be interpreted as angle theta and radius r rather than as x and y. This could be used, for example, to place labels on a 2D plot in polar coordinates or a 3D plot in cylindrical coordinates.

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 **set arrow** ... **rto**). In most cases, the given value serves as difference to the first position. If the given coordinate belongs to a log-scaled axis, a relative value is interpreted as multiplier. 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 **timefmt** format string. See **set xdata (p. 213)** and **set timefmt (p. 208)**. **Gnuplot** will also accept an integer expression, which will be interpreted as seconds relative to 1 January 1970.

Datastrings

Data files may contain string data consisting 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 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 (i.e. the column heading) 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) with lines title columnhead(2)
```

Another example:

```
plot for [i=2:6] 'datafile' using i title "Results for ".columnhead(i)
```

This use of column headings is automated by **set datafile columnheaders** or **set key autotitle columnhead**. See **labels (p. 75)**, **using xticlabels (p. 117)**, **plot title (p. 123)**, **using (p. 115)**, **key autotitle (p. 159)**.

Enhanced text mode

Many terminal types support an enhanced text mode in which additional formatting information is embedded in the text string. For example, "x^2" will write x-squared as we are used to seeing it, with a superscript 2. This mode is selected by default when you set the terminal, but may be toggled afterward using "set termoption [no]enhanced", or by marking individual strings as in "set label 'x_2' noenhanced".

Enhanced Text Control Codes			
Control	Example	Result	Explanation
`^`	`a^x`	a^x	superscript
`_`	`a_x`	a_x	subscript
`@`	`a@^b_{cd}`	a^b_{cd}	phantom box (occupies no width)
`&`	`d&{space}b`	d␣␣␣␣␣b	inserts space of specified length
`~`	`~a{.8-}`	ã	overprints '-' on 'a', raised by .8 times the current fontsize
	`{/Times abc}`	abc	print abc in font Times at current size
	`{/Times*2 abc}`	abc	print abc in font Times at twice current size
	`{/Times:Italic abc}`	*abc*	print abc in font Times with style italic
	`{/Arial:Bold=20 abc}`	**abc**	print abc in boldface Arial font size 20
`\U+`	`\U+221E`	∞	Unicode point U+221E INFINITY

The markup control characters act on the following single character or bracketed clause. The bracketed clause may contain a string of characters with no additional markup, e.g. 2^{10}, or it may contain additional markup that changes font properties. Font specifiers MUST be preceded by a '/' character that immediately follows the opening '{'. If a font name contains spaces it must be enclosed in single or double quotes.

Examples: The first example illustrates nesting one bracketed clause inside another to produce a boldface A with an italic subscript i, all in the current font. If the clause introduced by :Normal were omitted the subscript would be both italic and boldface. The second example illustrates the same markup applied to font "Times New Roman" at 20 point size.

```
{/:Bold A_{/:Normal{/:Italic i}}}
{/"Times New Roman":Bold=20 A_{/:Normal{/:Italic i}}}
```

The phantom box is useful for a@^b_c to align superscripts and subscripts but does not work well for overwriting an accent on a letter. For the latter, it is much better to use an encoding (e.g. iso_8859_1 or utf8) that contains a large variety of letters with accents or other diacritical marks. See **set encoding (p. 146)**. Since the box is non-spacing, it is sensible to put the shorter of the subscript or superscript in the box (that is, after the @).

Space equal in length to a string can be inserted using the '&' character. Thus

```
'abc&{def}ghi'
```

would produce

```
'abc   ghi'.
```

The '~ ' character causes the next character or bracketed text to be overprinted by the following character or bracketed text. The second text will be horizontally centered on the first. Thus '~ a/' will result in an 'a' with a slash through it. You can also shift the second text vertically by preceding the second text with a number, which will define the fraction of the current fontsize by which the text will be raised or lowered. In this case the number and text must be enclosed in brackets because more than one character is necessary. If the overprinted text begins with a number, put a space between the vertical offset and the text ('~ {abc}{.5 000}'); otherwise no space is needed ('~ {abc}{.5 — }'). You can change the font for one or both strings ('~ a{.5 /*.2 o}' — an 'a' with a one-fifth-size 'o' on top — and the space between the number and the slash is necessary), but you can't change it after the beginning of the string. Neither can you use any other special syntax within either string. You can, of course, use control characters by escaping them (see below), such as '~ a{\^}'

You can escape control characters using \, e.g., \\, \{, and so on. See **escape sequences (p. 35)** below.

Note that strings in double-quotes are parsed differently than those enclosed in single-quotes. The major difference is that backslashes may need to be doubled when in double-quoted strings.

The file "ps_guide.ps" in the /docs/psdoc subdirectory of the gnuplot source distribution contains more examples of the enhanced syntax, as does the demo **enhanced_utf8.dem**

Escape sequences

The backslash character \ is used to escape single byte character codes or Unicode entry points.

The form \ooo (where ooo is a 3 character octal value) can be used to index a known character code in a specific font encoding. For example the Adobe Symbol font uses a custom encoding in which octal 245 represents the infinity symbol. You could embed this in an enhanced text string by giving the font name and the character code "{/Symbol \245}". This is mostly useful for the PostScript terminal, which cannot easily handle UTF-8 encoding.

You can specify a character by its Unicode code point as \U+hhhh, where hhhh is the 4 or 5 character hexadecimal code point. For example the code point for the infinity symbol is \U+221E. This will be converted to a UTF-8 byte sequence on output if appropriate. In a UTF-8 environment this mechanism is not needed for printable special characters since they are handled in a text string like any other character. However it is useful for combining forms or supplemental diacritical marks (e.g. an arrow over a letter to represent a vector). See **set encoding (p. 146)**, **utf8 (p. 146)**, and the `online unicode demo.`

Environment

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

GNUTERM, if defined, is used to set the terminal type on start-up. Starting with version 5.2 the entire string in GNUTERM is passed to "set term" so that terminal options may be included. E.g.

```
GNUTERM="postscript eps color size 5in, 3in"
```

This can be overridden by the ~ /.gnuplot (or equivalent) start-up file (see **startup (p. 55)**) and of course by later explicit **set term** commands.

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 VMS system help library.

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 MS-DOS, Windows and OS/2, GNUPLOT is used. On Windows, the NT-specific variable USERPROFILE is also tried. VMS, SYS$LOGIN: is used. Type **help startup**.

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

On Unix, SHELL is used for the **shell** command. On MS-DOS and OS/2, COMSPEC is used for the **shell** command.

FIT_SCRIPT may be used to specify a **gnuplot** command to be executed when a fit is interrupted — see **fit (p. 90)**. **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 platforms. The contents of GNUPLOT_LIB are appended to the **loadpath** variable, but not saved with the **save** 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.

GNUPLOT_PS_DIR is used by the postscript driver to search for external prologue files. Depending on the build process, gnuplot contains either a built-in copy of those files or a default hardcoded path. You can use this variable have the postscript terminal use custom prologue files rather than the default files. See **postscript prologue (p. 279)**.

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.

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 always rounds down (-5/2 yields -3) or always rounds toward zero (-5/2 yields -2).

The integer expression "1/0" may be used to generate an "undefined" flag, which causes a point to be ignored. Or you can use the pre-defined variable NaN to achieve the same result. See **using (p. 115)** for an example.

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.

Complex arithmetic

Arithmetic operations and most built-in functions support the use of complex arguments. Complex constants are expressed as {<real>,<imag>}, where <real> and <imag> must be numerical constants. Thus {0,1} represents 'i'. The real and imaginary components of complex value x can be extracted as real(x) and imag(x). The modulus is given by abs(x).

Gnuplot's standard 2D and 3D plot styles can plot only real values; if you need to plot a complex-valued function f(x) with non-zero imaginary components you must choose between plotting real(f(x)) or abs(f(x)). For examples of representing complex values using color, see the `complex trigonometric function demos (complex_trig.dem)`

Constants

Integer constants are interpreted via the C library routine strtoll(). This means that constants beginning with "0" are interpreted as octal, and constants beginning with "0x" or "0X" are interpreted as hexadecimal.

Floating point constants are interpreted via the C library routine atof().

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.

String constants consist of any sequence of characters enclosed either in single quotes or double quotes. The distinction between single and double quotes is important. See **quotes (p. 59)**.

Examples:

```
1 -10 0xffaabb        # integer constants
1.0 -10. 1e1 3.5e-1   # floating point constants
{1.2, -3.4}           # complex constant
"Line 1\nLine 2"      # string constant (\n is expanded to newline)
'123\n456'            # string constant (\ and n are ordinary characters)
```

Functions

Arguments to math functions in **gnuplot** can be integer, real, or complex unless otherwise noted. Functions that accept or return angles (e.g. sin(x)) treat angle values as radians, but this may be changed to degrees using the command **set angles**.

Math library functions		
Function	Arguments	Returns
abs(x)	any	absolute value of x, $\lvert x\rvert$; same type
abs(x)	complex	length of x, $\sqrt{\mathrm{real}(x)^2 + \mathrm{imag}(x)^2}$
acos(x)	any	$\cos^{-1} x$ (inverse cosine)
acosh(x)	any	$\cosh^{-1} x$ (inverse hyperbolic cosine) in radians
airy(x)	any	Airy function Ai(x)
arg(x)	complex	the phase of x
asin(x)	any	$\sin^{-1} x$ (inverse sin)
asinh(x)	any	$\sinh^{-1} x$ (inverse hyperbolic sin) in radians
atan(x)	any	$\tan^{-1} x$ (inverse tangent)
atan2(y,x)	int or real	$\tan^{-1}(y/x)$ (inverse tangent)
atanh(x)	any	$\tanh^{-1} x$ (inverse hyperbolic tangent) in radians
EllipticK(k)	real k ∈ (-1:1)	$K(k)$ complete elliptic integral of the first kind
EllipticE(k)	real k ∈ [-1:1]	$E(k)$ complete elliptic integral of the second kind
EllipticPi(n,k)	real n<1, real k ∈ (-1:1)	$\Pi(n, k)$ complete elliptic integral of the third kind
besj0(x)	int or real	J_0 Bessel function of x in radians
besj1(x)	int or real	J_1 Bessel function of x in radians
besjn(n,x)	int, real	J_n Bessel function of x in radians
besy0(x)	int or real	Y_0 Bessel function of x in radians
besy1(x)	int or real	Y_1 Bessel function of x in radians
besyn(n,x)	int, real	Y_n Bessel function of x in radians
besi0(x)	real	Modified Bessel function of order 0, x in radians
besi1(x)	real	Modified Bessel function of order 1, x in radians

Math library functions		
Function	Arguments	Returns
besin(n,x)	int, real	Modified Bessel function of order n, x in radians
ceil(x)	any	$\lceil x \rceil$, smallest integer not less than x (real part)
cos(x)	any	$\cos x$, cosine of x
cosh(x)	any	$\cosh x$, hyperbolic cosine of x in radians
erf(x)	any	erf(real(x)), error function of real(x)
erfc(x)	any	erfc(real(x)), 1.0 - error function of real(x)
exp(x)	any	e^x, exponential function of x
expint(n,x)	int $n \geq 0$, real $x \geq 0$	$E_n(x) = \int_1^\infty t^{-n} e^{-xt}\, dt$, exponential integral of x
floor(x)	any	$\lfloor x \rfloor$, largest integer not greater than x (real part)
gamma(x)	any	gamma(real(x)), gamma function of real(x)
ibeta(p,q,x)	any	ibeta(real(p,q,x)), ibeta function of real(p,q,x)
inverf(x)	any	inverse error function of real(x)
igamma(a,x)	any	igamma(real(a,x)), igamma function of real(a,x)
imag(x)	complex	imaginary part of x as a real number
invnorm(x)	any	inverse normal distribution function of real(x)
int(x)	real	integer part of x, truncated toward zero
lambertw(x)	real	Lambert W function
lgamma(x)	any	lgamma(real(x)), lgamma function of real(x)
log(x)	any	$\log_e x$, natural logarithm (base e) of x
log10(x)	any	$\log_{10} x$, logarithm (base 10) of x
norm(x)	any	normal distribution (Gaussian) function of real(x)
rand(x)	int	pseudo random number in the open interval (0:1)
real(x)	any	real part of x
sgn(x)	any	1 if $x > 0$, -1 if $x < 0$, 0 if $x = 0$. imag(x) ignored
sin(x)	any	$\sin x$, sine of x
sinh(x)	any	$\sinh x$, hyperbolic sine of x in radians
sqrt(x)	any	$\sqrt{x}$, square root of x
tan(x)	any	$\tan x$, tangent of x
tanh(x)	any	$\tanh x$, hyperbolic tangent of x in radians
voigt(x,y)	real	Voigt/Faddeeva function $\frac{y}{\pi} \int \frac{exp(-t^2)}{(x-t)^2+y^2} dt$ Note: voigt(x, y) = $real$(faddeeva($x + iy$))

Special functions from libcerf (only if available)		
Function	Arguments	Returns
cerf(z)	complex	complex error function
cdawson(z)	complex	complex extension of Dawson's integral $D(z) = \frac{\sqrt{\pi}}{2} e^{-z^2} erfi(z)$
faddeeva(z)	complex	rescaled complex error function $w(z) = e^{-z^2}\ erfc(-iz)$
erfi(x)	real	imaginary error function $erf(x) = -i * erf(ix)$
VP(x,σ,γ)	real	Voigt profile $VP(x, \sigma, \gamma) = \int_{-\infty}^{\infty} G(x'; \sigma) L(x - x'; \gamma) dx'$

String functions		
Function	Arguments	Returns
gprintf("format",x,...)	any	string result from applying gnuplot's format parser
sprintf("format",x,...)	multiple	string result from C-language sprintf
strlen("string")	string	number of characters in string
strstrt("string","key")	strings	int index of first character of substring "key"
substr("string",beg,end)	multiple	string "string"[beg:end]
strftime("timeformat",t)	any	string result from applying gnuplot's time parser
strptime("timeformat",s)	string	seconds since year 1970 as given in string s
system("command")	string	string containing output stream of shell command
trim(" string ")	string	string without leading or trailing whitespace
word("string",n)	string, int	returns the nth word in "string"
words("string")	string	returns the number of words in "string"

other **gnuplot** functions		
Function	Arguments	Returns
column(x)	int or string	column x during datafile manipulation.
columnhead(x)	int	string containing first entry of column x in datafile.
exists("X")	string	returns 1 if a variable named X is defined, 0 otherwise.
hsv2rgb(h,s,v)	h,s,v $\in$ [0:1]	24bit RGB color value.
palette(z)	double	RGB palette color mapped to z.
stringcolumn(x)	int or string	content of column x as a string.
timecolumn(N,"timeformat")	int, string	time data from column N during data input.
tm_hour(t)	time in sec	the hour (0..23)
tm_mday(t)	time in sec	the day of the month (1..31)
tm_min(t)	time in sec	the minute (0..59)
tm_mon(t)	time in sec	the month (0..11)
tm_sec(t)	time in sec	the second (0..59)
tm_wday(t)	time in sec	the day of the week (Sun..Sat) as (0..6)
tm_week(t)	time in sec	week of year in ISO8601 "week date" system (1..53)
tm_yday(t)	time in sec	the day of the year (0..365)
tm_year(t)	time in sec	the year
time(x)	any	the current system time in seconds
valid(x)	int	test validity of column(x) during datafile manip.
value("name")	string	returns the value of the named variable.
voxel(x,y,z)	real	value of the active grid voxel containing point (x,y,z)

Elliptic integrals

The **EllipticK(k)** function returns the complete elliptic integral of the first kind, i.e. the definite integral between 0 and pi/2 of the function **(1-(k*sin(p))**2)**(-0.5)**. The domain of **k** is -1 to 1 (exclusive).

The **EllipticE(k)** function returns the complete elliptic integral of the second kind, i.e. the definite integral between 0 and pi/2 of the function **(1-(k*sin(p))**2)**0.5**. The domain of **k** is -1 to 1 (inclusive).

The **EllipticPi(n,k)** function returns the complete elliptic integral of the third kind, i.e. the definite integral between 0 and pi/2 of the function **(1-(k*sin(p))**2)**(-0.5)/(1-n*sin(p)**2)**. The parameter **n** must be less than 1, while **k** must lie between -1 and 1 (exclusive). Note that by definition EllipticPi(0,k) == EllipticK(k) for all possible values of **k**.

Random number generator

The function **rand()** produces a sequence of pseudo-random numbers between 0 and 1 using an algorithm from P. L'Ecuyer and S. Cote, "Implementing a random number package with splitting facilities", ACM

Transactions on Mathematical Software, 17:98-111 (1991).

```
rand(0)      returns a pseudo random number in the open 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 integer 0 < x < 2^31-1 sets both internal seeds
             to x.
rand({x,y})  for integer 0 < x,y < 2^31-1 sets seed1 to x and
             seed2 to y.
```

Value

B = value("A") is effectively the same as B = A, where A is the name of a user-defined variable. This is useful when the name of the variable is itself held in a string variable. See **user-defined variables (p. 43)**. It also allows you to read the name of a variable from a data file. If the argument is a numerical expression, value() returns the value of that expression. If the argument is a string that does not correspond to a currently defined variable, value() returns NaN.

Counting and extracting words

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

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

The **word** and **words** functions provide limited support for quoted strings, both single and double quotes can be used:

```
print words("\"double quotes\" or 'single quotes'")   # 3
```

A starting quote must either be preceded by a white space, or start the string. This means that apostrophes in the middle or at the end of words are considered as parts of the respective word:

```
print words("Alexis' phone doesn't work") # 4
```

Escaping quote characters is not supported. If you want to keep certain quotes, the respective section must be surrounded by the other kind of quotes:

```
s = "Keep \"'single quotes'\" or '\"double quotes\"'"
print word(s, 2) # 'single quotes'
print word(s, 4) # "double quotes"
```

Note, that in this last example the escaped quotes are necessary only for the string definition.

trim(" padded string ") returns the original string stripped of leading and trailing whitespace. This is useful for string comparisons of input data fields that may contain extra whitespace. For example

```
plot FOO using 1:( trim(strcol(3)) eq "A" ? $2 : NaN )
```

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.

Unary

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

<table>
<tr><th colspan="3">Unary Operators</th></tr>
<tr><th>Symbol</th><th>Example</th><th>Explanation</th></tr>
<tr><td>-</td><td>-a</td><td>unary minus</td></tr>
<tr><td>+</td><td>+a</td><td>unary plus (no-operation)</td></tr>
<tr><td>~</td><td>~a</td><td>* one's complement</td></tr>
<tr><td>!</td><td>!a</td><td>* logical negation</td></tr>
<tr><td>!</td><td>a!</td><td>* factorial</td></tr>
<tr><td>$</td><td>$3</td><td>* call arg/column during 'using' manipulation</td></tr>
<tr><td>|</td><td>|A|</td><td>cardinality of array A</td></tr>
</table>

(*) 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 an integer when N! is sufficiently small (N <= 20 for 64-bit integers). It returns a floating point approximation for larger values of N.

This operator returns the number of elements |A| when applied to array A. It returns the number of data lines |$DATA| when applied to datablock $DATA.

Binary

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

<table>
<tr><th colspan="3">Binary Operators</th></tr>
<tr><th>Symbol</th><th>Example</th><th>Explanation</th></tr>
<tr><td>**</td><td>a**b</td><td>exponentiation</td></tr>
<tr><td>*</td><td>a*b</td><td>multiplication</td></tr>
<tr><td>/</td><td>a/b</td><td>division</td></tr>
<tr><td>%</td><td>a%b</td><td>* modulo</td></tr>
<tr><td>+</td><td>a+b</td><td>addition</td></tr>
<tr><td>-</td><td>a-b</td><td>subtraction</td></tr>
<tr><td>==</td><td>a==b</td><td>equality</td></tr>
<tr><td>!=</td><td>a!=b</td><td>inequality</td></tr>
<tr><td><</td><td>a<b</td><td>less than</td></tr>
<tr><td><=</td><td>a<=b</td><td>less than or equal to</td></tr>
<tr><td>></td><td>a>b</td><td>greater than</td></tr>
<tr><td>>=</td><td>a>=b</td><td>greater than or equal to</td></tr>
<tr><td><<</td><td>0xff<<1</td><td>left shift unsigned</td></tr>
<tr><td>>></td><td>0xff>>1</td><td>right shift unsigned</td></tr>
<tr><td>&</td><td>a&b</td><td>* bitwise AND</td></tr>
<tr><td>^</td><td>a^b</td><td>* bitwise exclusive OR</td></tr>
<tr><td>|</td><td>a|b</td><td>* bitwise inclusive OR</td></tr>
<tr><td>&&</td><td>a&&b</td><td>* logical AND</td></tr>
<tr><td>||</td><td>a||b</td><td>* logical OR</td></tr>
<tr><td>=</td><td>a = b</td><td>assignment</td></tr>
<tr><td>,</td><td>(a,b)</td><td>serial evaluation</td></tr>
<tr><td>.</td><td>A.B</td><td>string concatenation</td></tr>
<tr><td>eq</td><td>A eq B</td><td>string equality</td></tr>
<tr><td>ne</td><td>A ne B</td><td>string inequality</td></tr>
</table>

(*) 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.

Serial evaluation occurs only in parentheses and is guaranteed to proceed in left to right order. The value of the rightmost subexpression is returned.

Ternary

There is a single ternary operator:

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 )
```

For an explanation of the **using** syntax, please see **plot datafile using (p. 115)**.

Summation

A summation expression has the form

```
sum [<var> = <start> : <end>] <expression>
```

<var> is treated as an integer variable that takes on successive integral values from <start> to <end>. For each of these, the current value of <expression> is added to a running total whose final value becomes the value of the summation expression. Examples:

```
print sum [i=1:10] i
    55.
# Equivalent to plot 'data' using 1:($2+$3+$4+$5+$6+...)
plot 'data' using 1 : (sum [col=2:MAXCOL] column(col))
```

It is not necessary that <expression> contain the variable <var>. Although <start> and <end> can be specified as variables or expressions, their value cannot be changed dynamically as a side-effect of carrying out the summation. If <end> is less than <start> then the value of the summation is zero.

Gnuplot-defined variables

Gnuplot maintains a number of read-only variables that reflect the current internal state of the program and the most recent plot. These variables begin with the prefix "GPVAL_". Examples include GPVAL_TERM, GPVAL_X_MIN, GPVAL_X_MAX, GPVAL_Y_MIN. Type **show variables all** to display the complete list and current values. Values related to axes parameters (ranges, log base) are values used during the last plot, not those currently **set**.

Example: To calculate the fractional screen coordinates of the point [X,Y]

```
GRAPH_X = (X - GPVAL_X_MIN) / (GPVAL_X_MAX - GPVAL_X_MIN)
GRAPH_Y = (Y - GPVAL_Y_MIN) / (GPVAL_Y_MAX - GPVAL_Y_MIN)
SCREEN_X = GPVAL_TERM_XMIN + GRAPH_X * (GPVAL_TERM_XMAX - GPVAL_TERM_XMIN)
SCREEN_Y = GPVAL_TERM_YMIN + GRAPH_Y * (GPVAL_TERM_YMAX - GPVAL_TERM_YMIN)
FRAC_X = SCREEN_X * GPVAL_TERM_SCALE / GPVAL_TERM_XSIZE
FRAC_Y = SCREEN_Y * GPVAL_TERM_SCALE / GPVAL_TERM_YSIZE
```

The read-only variable GPVAL_ERRNO is set to a non-zero value if any gnuplot command terminates early due to an error. The most recent error message is stored in the string variable GPVAL_ERRMSG. Both GPVAL_ERRNO and GPVAL_ERRMSG can be cleared using the command **reset errors**.

Interactive terminals with **mouse** functionality maintain read-only variables with the prefix "MOUSE_". See **mouse variables (p. 54)** for details.

The **fit** mechanism uses several variables with names that begin "FIT_". It is safest to avoid using such names. When using **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 (p. 90)** and **set fit (p. 147)** for details.

See **user-defined variables (p. 43)**, **reset errors (p. 128)**, **mouse variables (p. 54)**, and **fit (p. 90)**.

User-defined variables and functions

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

User-defined function syntax:

```
<func-name>( <dummy1> {,<dummy2>} ... {,<dummy12>} ) = <expression>
```

where <expression> is defined in terms of <dummy1> through <dummy12>.

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 variables **pi** (3.14159...) and **NaN** (IEEE "Not a Number") are already defined. You can redefine these to something else if you really need to. The original values can be recovered by setting:

```
NaN = GPVAL_NaN
pi  = GPVAL_pi
```

Other variables may be defined under various gnuplot operations like mousing in interactive terminals or fitting; see **gnuplot-defined variables (p. 43)** for details.

You can check for existence of a given variable V by the exists("V") expression. For example

```
a = 10
if (exists("a")) print "a is defined"
if (!exists("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 "_".

Each function definition is made available as a special string-valued variable with the prefix 'GPFUN_'.

Example:

```
set label GPFUN_sinc at graph .05,.95
```

See **show functions (p. 153)**, **functions (p. 119)**, **gnuplot-defined variables (p. 43)**, **macros (p. 57)**, **value (p. 40)**.

Arrays

Arrays are implemented as indexed lists of user variables. The elements in an array are not limited to a single type of variable. Arrays must be created explicitly before being referenced. The size of an array cannot be changed after creation. All elements are initially undefined. In most places an array element can be used instead of a named user variable.

The cardinality (number of elements) of array A is given by the expression |A|.

Example:

```
array A[6]
A[1] = 1
A[2] = 2.0
A[3] = {3.0, 3.0}
A[4] = "four"
A[6] = A[2]**3
array B[6] = [ 1, 2.0, A[3], "four", , B[2]**3 ]

do for [i=1:6] { print A[i], B[i] }
    1 1
    2.0 2.0
    {3.0, 3.0} {3.0, 3.0}
    four four
    <undefined> <undefined>
    8.0 8.0
```

Note: Arrays and variables share the same namespace. For example, assignment of a string variable named FOO will destroy any previously created array with name FOO.

The name of an array can be used in a **plot**, **splot**, **fit**, or **stats** command. This is equivalent to providing a file in which column 1 holds the array index (from 1 to size), column 2 holds the value of real(A[i]) and column 3 holds the value of imag(A[i]).

Example:

```
array A[200]
do for [i=1:200] { A[i] = sin(i * pi/100.) }
plot A title "sin(x) in centiradians"
```

When plotting the imaginary component of complex array values, it may be referenced either as imag(A[$1]) or as $3. These two commands are equivalent

```
plot A using (real(A[$1])) : (imag(A[$1]))
plot A using 2:3
```

Fonts

Gnuplot does not provide any fonts of its own. It relies on external font handling, the details of which unfortunately vary from one terminal type to another. Brief documentation of font mechanisms that apply to more than one terminal type is given here. For information on font use by other individual terminals, see the documentation for that terminal.

Although it is possible to include non-alphabetic symbols by temporarily switching to a special font, e.g. the Adobe Symbol font, the preferred method is now to choose UTF-8 encoding and treat the symbol like any other character. Alternatively you can specify the unicode entry point for the desired symbol as an escape sequence in enhanced text mode. See **encoding (p. 146)**, **unicode (p. 35)**, **locale (p. 166)**, and **escape sequences (p. 35)**.

Cairo (pdfcairo, pngcairo, epscairo, wxt terminals)

These terminals find and access fonts using the external fontconfig tool set. Please see the `fontconfig user manual.`

It is usually sufficient in gnuplot to request a font by a generic name and size, letting fontconfig substitute a similar font if necessary. The following will probably all work:

```
set term pdfcairo font "sans,12"
set term pdfcairo font "Times,12"
set term pdfcairo font "Times-New-Roman,12"
```

Gd (png, gif, jpeg, sixel terminals)

Font handling for the png, gif, jpeg, and sixelgd terminals is done by the library libgd. Five basic fonts are provided directly by libgd. These are **tiny** (5x8 pixels), **small** (6x12 pixels), **medium**, (7x13 Bold), **large** (8x16) or **giant** (9x15 pixels). These fonts cannot be scaled or rotated. Use one of these keywords instead of the **font** keyword. E.g.

```
set term png tiny
```

On most systems libgd also provides access to Adobe Type 1 fonts (*.pfa) and TrueType fonts (*.ttf). You must give the name of the font file, not the name of the font inside it, in the form "<face> {,<pointsize>}". <face> is either the full pathname to the font file, or 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. For example, if GDFONTPATH contains **/usr/local/fonts/ttf:/usr/local/fonts/pfa** then the following pairs of commands are equivalent

```
set term png font "arial"
set term png font "/usr/local/fonts/ttf/arial.ttf"
set term png font "Helvetica"
set term png font "/usr/local/fonts/pfa/Helvetica.pfa"
```

To request a default font size at the same time:

```
set term png font "arial,11"
```

Both TrueType and Adobe Type 1 fonts are fully scalable and rotatable. If no specific font is requested in the "set term" command, gnuplot checks the environmental variable GNUPLOT_DEFAULT_GDFONT to see if there is a preferred default font.

Postscript (also encapsulated postscript *.eps)

PostScript font handling is done by the printer or viewing program. Gnuplot can create valid PostScript or encapsulated PostScript (*.eps) even if no fonts at all are installed on your computer. Gnuplot simply refers to the font by name in the output file, and assumes that the printer or viewing program will know how to find or approximate a font by that name.

All PostScript printers or viewers should know about the standard set of Adobe fonts **Times-Roman**, **Helvetica**, **Courier**, and **Symbol**. It is likely that many additional fonts are also available, but the specific set depends on your system or printer configuration. Gnuplot does not know or care about this; the output *.ps or *.eps files that it creates will simply refer to whatever font names you request.

Thus

```
set term postscript eps font "Times-Roman,12"
```

will produce output that is suitable for all printers and viewers.

On the other hand

```
set term postscript eps font "Garamond-Premier-Pro-Italic"
```

will produce an output file that contains valid PostScript, but since it refers to a specialized font, only some printers or viewers will be able to display the specific font that was requested. Most will substitute a different font.

However, it is possible to embed a specific font in the output file so that all printers will be able to use it. This requires that the a suitable font description file is available on your system. Note that some font files require specific licensing if they are to be embedded in this way. See **postscript fontfile (p. 278)** for more detailed description and examples.

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" or "canvas" is the entire area addressable by **gnuplot**. On a desktop it is a full window; on a plotter, it is a single sheet of paper; in svga mode it is the full monitor screen.

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 2D graph may have up to four labeled **axes**. The names of the four axes are "x" for the axis along the bottom border of the plot, "y" for the axis along the left border, "x2" for the top border, and "y2" for the right border. See **axes (p. 102)**.

A 3D graph may have up to three labeled **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 **set view**.

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 "block" of data is a set of consecutive records delimited by blank records. A line, when referred to in the context of a data file, is a subset of a block. Note that the term "data block" may also be used to refer to a named block inline data (see **datablocks (p. 47)**).

Inline data and datablocks

There are two mechanisms for embedding data into a stream of gnuplot commands. If the special filename '-' appears in a plot command, then the lines immediately following the plot command are interpreted as inline data. See **special-filenames (p. 113)**. Data provided in this way can only be used once, by the plot command it follows.

The second mechanism defines a named data block as a here-document. The named data is persistent and may be referred to by more than one plot command. Example:

```
$Mydata << EOD
11 22 33 first line of data
44 55 66 second line of data
# comments work just as in a data file
77 88 99
EOD
stats $Mydata using 1:3
plot $Mydata using 1:3 with points, $Mydata using 1:2 with impulses
```

Data block names must begin with a $ character, which distinguishes them from other types of persistent variables. The end-of-data delimiter (EOD in the example) may be any sequence of alphanumeric characters.

The storage associated with named data blocks can be released using **undefine** command. **undefine $*** frees all named data blocks at once.

Iteration

gnuplot supports command iteration and block-structured if/else/while/do constructs. See **if (p. 98)**, **while (p. 235)**, and **do (p. 89)**. Simple iteration is possible inside **plot** or **set** commands. See **plot for (p. 122)**. General iteration spanning multiple commands is possible using a block construct as shown below. For a related new feature, see the **summation (p. 42)** expression type. Here is an example using several of these new syntax features:

```
set multiplot layout 2,2
fourier(k, x) = sin(3./2*k)/k * 2./3*cos(k*x)
do for [power = 0:3] {
    TERMS = 10**power
    set title sprintf("%g term Fourier series",TERMS)
    plot 0.5 + sum [k=1:TERMS] fourier(k,x) notitle
}
unset multiplot
```

Iteration is controlled by an iteration specifier with syntax

```
for [<var> in "string of N elements"]
```

or

```
for [<var> = <start> : <end> { : <increment> }]
```

In the first case <var> is a string variable that successively evaluates to single-word substrings 1 to N of the string in the iteration specifier. In the second case <start>, <end>, and <increment> are integers or integer expressions.

With one exception, gnuplot variables are global. There is a single, persistent, list of active variables indexed by name. Assignment to a variable creates or replaces an entry in that list. The only way to remove a variable from that list is the **undefine** command.

The single exception to this is the variable used in an iteration specifier. The scope of the iteration variable is private to that iteration. You cannot permanently change the value of the iteration variable inside the iterated clause. If the iteration variable has a value prior to iteration, that value will be retained or restored at the end of the iteration. For example, the following commands will print 1 2 3 4 5 6 7 8 9 10 A.

```
i = "A"
do for [i=1:10] { print i; i=10; }
print i
```

Linetypes, colors, and styles

In older gnuplot versions, each terminal type provided a set of distinct "linetypes" that could differ in color, in thickness, in dot/dash pattern, or in some combination of color and dot/dash. These colors and patterns were not guaranteed to be consistent across different terminal types although most used the color sequence red/green/blue/magenta/cyan/yellow. You can select this old behaviour via the command **set colorsequence classic**, but by default gnuplot version 5 uses a terminal-independent sequence of 8 colors.

You can further customize the sequence of linetype properties interactively or in an initialization file. See **set linetype (p. 164)**. Several sample initialization files are provided in the distribution package.

The current linetype properties for a particular terminal can be previewed by issuing the **test** command after setting the terminal type.

Successive functions or datafiles plotted by a single command will be assigned successive linetypes in the current default sequence. You can override this for any individual function, datafile, or plot element by giving explicit line properties in the plot command.

Examples:

```
plot "foo", "bar"              # plot two files using linetypes 1, 2
plot sin(x) linetype 4         # use linetype color 4
```

In general, colors can be specified using named colors, rgb (red, green, blue) components, hsv (hue, saturation, value) components, or a coordinate along the current pm3d palette.

Examples:

```
plot sin(x) lt rgb "violet"      # one of gnuplot's named colors
plot sin(x) lt rgb "#FF00FF"     # explicit RGB triple in hexadecimal
plot sin(x) lt palette cb -45    # whatever color corresponds to -45
                                 # in the current cbrange of the palette
plot sin(x) lt palette frac 0.3  # fractional value along the palette
```

See **colorspec (p. 49)**, **show colornames (p. 139)**, **hsv (p. 39)**, **set palette (p. 180)**, **cbrange (p. 225)**. See also **set monochrome (p. 168)**.

Linetypes also have an associated dot-dash pattern although not all terminal types are capable of using it. Gnuplot version 5 allows you to specify the dot-dash pattern independent of the line color. See **dashtype (p. 50)**.

Colorspec

Many commands allow you to specify a linetype with an explicit color.

Syntax:

```
... {linecolor | lc} {"colorname" | <colorspec> | <n>}
... {textcolor | tc} {<colorspec> | {linetype | lt} <n>}
... {fillcolor | fc} {<colorspec> | linetype <n> | linestyle <n>}
```

where <colorspec> has one of the following forms:

```
rgbcolor "colorname"    # e.g. "blue"
rgbcolor "0xRRGGBB"     # string containing hexadecimal constant
rgbcolor "0xAARRGGBB"   # string containing hexadecimal constant
rgbcolor "#RRGGBB"      # string containing hexadecimal in x11 format
rgbcolor "#AARRGGBB"    # string containing hexadecimal in x11 format
rgbcolor <integer val>  # integer value representing AARRGGBB
rgbcolor variable       # integer value is read from input file
palette frac <val>      # <val> runs from 0 to 1
palette cb <value>      # <val> lies within cbrange
palette z
variable                # color index is read from input file
bgnd                    # background color
black
```

The "<n>" is the linetype number the color of which is used, see **test (p. 233)**.

"colorname" refers to one of the color names built in to gnuplot. For a list of the available names, see **show colornames (p. 139)**.

Hexadecimal constants can be given in quotes as "#RRGGBB" or "0xRRGGBB", where RRGGBB represents the red, green, and blue components of the color and must be between 00 and FF. For example, magenta = full-scale red + full-scale blue could be represented by "0xFF00FF", which is the hexadecimal representation of (255 << 16) + (0 << 8) + (255).

"#AARRGGBB" represents an RGB color with an alpha channel (transparency) value in the high bits. An alpha value of 0 represents a fully opaque color; i.e., "#00RRGGBB" is the same as "#RRGGBB". An alpha value of 255 (FF) represents full transparency.

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 **set cbrange (p. 225)**. See also **set colorbox (p. 139)**. 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. It also allows coloring 2D plots by palette values read from an extra column of data (not all 2D plot styles allow an extra column). There are two special color specifiers: **bgnd** for background color and **black**.

Background color

Most terminals allow you to set an explicit background color for the plot. The special linetype **bgnd** will draw in this color, and **bgnd** is also recognized as a color. Examples:

```
# This will erase a section of the canvas by writing over it in the
# background color
set term wxt background rgb "gray75"
set object 1 rectangle from x0,y0 to x1,y1 fillstyle solid fillcolor bgnd
# This will draw an "invisible" line along the x axis
plot 0 lt bgnd
```

Linecolor variable

lc variable tells the program to use the value read from one column of the input data as a linetype index, and use the color belonging to that linetype. This requires a corresponding additional column in the **using** specifier. Text colors can be set similarly using **tc variable**.

Examples:

```
# Use the third column of data to assign colors to individual points
plot 'data' using 1:2:3 with points lc variable

# A single data file may contain multiple sets of data, separated by two
# blank lines.  Each data set is assigned as index value (see `index`)
# that can be retrieved via the `using` specifier `column(-2)`.
# See `pseudocolumns`.  This example uses to value in column -2 to
# draw each data set in a different line color.
plot 'data' using 1:2:(column(-2)) with lines lc variable
```

Rgbcolor variable

You can assign a separate color for each data point, line segment, or label in your plot. **lc rgbcolor variable** tells the program to read RGB color information for each line in the data file. This requires a corresponding additional column in the **using** specifier. The extra column is interpreted as a 24-bit packed RGB triple. If the value is provided directly in the data file it is easiest to give it as a hexadecimal value (see **rgbcolor** **(p. 49)**). Alternatively, the **using** specifier can contain an expression that evaluates to a 24-bit RGB color as in the example below. Text colors are similarly set using **tc rgbcolor variable**.

Example:

```
# Place colored points in 3D at the x,y,z coordinates corresponding to
# their red, green, and blue components
rgb(r,g,b) = 65536 * int(r) + 256 * int(g) + int(b)
splot "data" using 1:2:3:(rgb($1,$2,$3)) with points lc rgb variable
```

Dashtype

In gnuplot version 5 the dash pattern (**dashtype**) is a separate property associated with each line, analogous to **linecolor** or **linewidth**. It is not necessary to place the current terminal in a special mode just to draw dashed lines. I.e. the command **set term <termname> {solid|dashed}** is now ignored. If backwards compatibility with old scripts written for version 4 is required, the following lines can be used instead:

```
if (GPVAL_VERSION >= 5.0) set for [i=1:9] linetype i dashtype i
if (GPVAL_VERSION < 5.0) set termoption dashed
```

All lines have the property **dashtype solid** unless you specify otherwise. You can change the default for a particular linetype using the command **set linetype** so that it affects all subsequent commands, or you can include the desired dashtype as part of the **plot** or other command.

Syntax:

```
dashtype N          # predefined dashtype invoked by number
dashtype "pattern"  # string containing a combination of the characters
                    # dot (.) hyphen (-) underscore(_) and space.
dashtype (s1,e1,s2,e2,s3,e3,s4,e4)  # dash pattern specified by 1 to 4
                    # numerical pairs <solid length>, <emptyspace length>
```

Example:

```
# Two functions using linetype 1 but distinguished by dashtype
plot f1(x) with lines lt 1 dt solid, f2(x) with lines lt 1 dt 3
```

Some terminals support user-defined dash patterns in addition to whatever set of predefined dash patterns they offer.

Examples:

```
plot f(x) dt 3            # use terminal-specific dash pattern 3
plot f(x) dt ".. "        # construct a dash pattern on the spot
plot f(x) dt (2,5,2,15)   # numerical representation of the same pattern
set dashtype 11 (2,4,4,7) # define new dashtype to be called by index
plot f(x) dt 11           # plot using our new dashtype
```

If you specify a dash pattern using a string the program will convert this to a sequence of <solid>,<empty> pairs. Dot "." becomes (2,5), dash "-" becomes (10,10), underscore "_" becomes (20,10), and each space character " " adds 10 to the previous <empty> value. The command **show dashtype** will show both the original string and the converted numerical sequence.

Linestyles vs linetypes

A **linestyle** is a temporary association of properties linecolor, linewidth, dashtype, and pointtype. It is defined 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. In other words, it is just like a linetype except for its lifetime. Whereas **linetypes** are permanent (they last until you explicitly redefine them), **linestyles** last until the next reset of the graphics state.

Examples:

```
# define a new line style with terminal-independent color cyan,
# linewidth 3, and associated point type 6 (a circle 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
```

Layers

A gnuplot plot is built up by drawing its various components in a fixed order. This order can be modified by assigning some components to a specific layer using the keywords **behind**, **back**, or **front**. For example, to replace the background color of the plot area you could define a colored rectangle with the attribute **behind**.

```
set object 1 rectangle from graph 0,0 to graph 1,1 fc rgb "gray" behind
```

The order of drawing is

```
behind
back
the plot itself
the plot legend (‘key‘)
front
```

Within each layer elements are drawn in the order

```
grid, axis, and border elements
pixmaps in numerical order
objects (rectangles, circles, ellipses, polygons) in numerical order
labels in numerical order
arrows in numerical order
```

In the case of multiple plots on a single page (multiplot mode) this order applies separately to each component plot, not to the multiplot as a whole.

Mouse input

Many terminals allow interaction with the current plot using the mouse. Some 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 **bind (p. 52)** and **mouse variables (p. 54)**. See also the command **set mouse (p. 169)**.

Bind

Syntax:

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

The **bind** 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 **bind** is only available if gnuplot was compiled with **mouse** support and it is used by all mouse-capable terminals. A user-specified binding supersedes any builtin bindings, except that <space> and 'q' cannot normally be rebound. For an exception, see **bind space (p. 53)**.

Only mouse button 1 can be bound, and only for 2D plots.

You get the list of all hotkeys by typing **show bind** or **bind** or by typing the hotkey 'h' in the graph window.

Key bindings are restored to their default state by **reset bind**.

Note that multikey-bindings with modifiers must be given in quotes.

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.

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
show bind              # show all bindings
```

- remove bindings:

```
bind "ctrl-alt-a" ""   # removes binding for ctrl-alt-a
                         (note that builtins cannot be removed)
reset 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 noraise; 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, shift (only valid for Button1 Button2 Button3)
```

List of supported special keys:

```
"BackSpace", "Tab", "Linefeed", "Clear", "Return", "Pause", "Scroll_Lock",
"Sys_Req", "Escape", "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"
```

The following are window events rather than actual keys

```
"Button1" "Button2" "Button3" "Close"
```

See also help for **mouse (p. 169)**.

Bind space

If gnuplot was built with configuration option –enable-raise-console, then typing <space> in the plot window raises gnuplot's command window. This hotkey can be changed to ctrl-space by starting gnuplot as 'gnuplot -ctrlq', or by setting the XResource 'gnuplot*ctrlq'. See **x11 command-line-options (p. 298)**.

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 (exists("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.

Note that after a zoom by mouse, you can read the new ranges as GPVAL_X_MIN, GPVAL_X_MAX, GPVAL_Y_MIN, and GPVAL_Y_MAX, see **gnuplot-defined variables (p. 43)**.

Persist

Many gnuplot terminals (aqua, pm, qt, x11, windows, wxt, ...) open separate display windows on the screen into which plots are drawn. The **persist** option tells gnuplot to leave these windows open when the main program exits. It has no effect on non-interactive terminal output. For example if you issue the command

```
gnuplot -persist -e 'plot [-5:5] sinh(x)'
```

gnuplot will open a display window, draw the plot into it, and then exit, leaving the display window containing the plot on the screen. You can also specify **persist** or **nopersist** when you set a new terminal.

```
set term qt persist size 700,500
```

Depending on the terminal type, some mousing operations may still be possible in the persistent window. However operations like zoom/unzoom that require redrawing the plot are not possible because the main program has exited. If you want to leave a plot window open and fully mouseable after creating the plot, for example when running gnuplot from a script file rather than interactively, see **pause mouse close (p. 101)**.

Plotting

There are four **gnuplot** commands which actually create a plot: **plot**, **splot**, **replot**, and **refresh**. Other commands control the layout, style, and content of the plot that will eventually be created. **plot** generates 2D plots. **splot** generates 3D plots (actually 2D projections, of course). **replot** reexecutes the previous **plot** or **splot** command. **refresh** is similar to **replot** but it reuses any previously stored data rather than rereading data from a file or input stream.

Each time you issue one of these four commands it will redraw the screen or generate a new page of output containing all of the currently defined axes, labels, titles, and all of the various functions or data sources listed in the original plot command. If instead you need to place several complete plots next to each other on the same page, e.g. to make a panel of sub-figures or to inset a small plot inside a larger plot, use the command **set multiplot** to suppress generation of a new page for each plot command.

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

plot operates in either rectangular or polar coordinates – see **set polar (p. 192)**. **splot** operates in Cartesian coordinates, but will accept azimuthal or cylindrical coordinates on input. See **set mapping (p. 166)**. **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 labeling of each axis. Some commands have the name of an axis built into their names, such as **set xlabel**. 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 2D graphs.

splot can plot surfaces and contours in addition to points and/or lines. See **set isosamples (p. 156)** for information about defining the grid for a 3D function. See **splot datafile (p. 226)** for information about the requisite file structure for 3D data. For contours see **set contour (p. 140)**, **set cntrlabel (p. 136)**, and **set cntrparam (p. 137)**.

In **splot**, control over the scales and labels of the axes are the same as with **plot** except that there is also a z axis and labeling the x2 and y2 axes is possible only for pseudo-2D plots created using **set view map**.

Plugins

The set of functions available for plotting or for evaluating expressions can be extended through a plugin mechanism that imports executable functions from a shared library. For example, gnuplot versions through 5.4 do not provide a built-in implementation of the upper incomplete gamma function Q(a,x).
$Q(a,x) = \frac{1}{\Gamma(x)} \int_x^\infty t^{a-1} e^{-t} dt$.

You could define an approximation directly in gnuplot like this:

```
Q(a,x) = 1. - igamma(a,x)
```

However this has inherently limited precision as the value of Q approaches 1. If you need a more accurate implementation, it would be better to provide one via a plugin (see below). Once imported, the function can be used just as any other built-in or user-defined function in gnuplot. See **import (p. 99)**.

The gnuplot distribution includes source code and instructions for creating a plugin library in the directory demo/plugin. You can modify the simple example file **demo_plugin.c** by replacing one or more of the toy example functions with an implementation of the function you are interested in. This could include invocation of functions from an external mathematical library.

The demo/plugin directory also contains source for a plugin that implements Q(a,x). As noted above, this plugin allows earlier versions of gnuplot to provide the same function **uigamma** as the current development version.

```
import Q(a,x) from "uigamma_plugin"
uigamma(a,x) = ((x<1 || x<a) ? 1.0-igamma(a,x) : Q(a,x))
```

Start-up (initialization)

When gnuplot is run, it first looks for a system-wide initialization file **gnuplotrc**. The location of this file is determined when the program is built and is reported by **show loadpath**. The program then looks in the user's HOME directory for a file called **.gnuplot** on Unix-like systems or **GNUPLOT.INI** on other systems. (OS/2 will look for it in the directory named in the environment variable **GNUPLOT**; Windows will use **APPDATA**). Note: The program can be configured to look first in the current directory, but this is not recommended because it is bad security practice.

String constants, string variables, and string functions

In addition to string constants, most gnuplot commands 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 ('.' character), 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
```

Substrings

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.

String operators

Three binary operators 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"
```

String functions

Gnuplot provides several built-in functions that operate on strings. General formatting functions: see **gprintf (p. 149) sprintf (p. 39)**. Time formatting functions: see **strftime (p. 39) strptime (p. 39)**. String manipulation: see **substr (p. 39) strstrt (p. 39) trim (p. 40) word (p. 40) words (p. 40)**.

String encoding

Gnuplot's built-in string manipulation functions are sensitive to utf-8 encoding (see **set encoding** **(p. 146)**). For example

```
utf8string = "αβγ"
strlen(utf8string) returns 3 (number of characters, not number of bytes)
utf8string[2:2] evaluates to "β"
strstrt(utf8string,"β") evaluates to 2
```

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).

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. Exit status of the system command is returned in variables GPVAL_SYSTEM_ERRNO and GPVAL_SYSTEM_ERRMSG.

CHANGE (differs from versions 4 through 5.2): Internal carriage-return ('\r') and newline ('\n') characters are not stripped from the input stream during substitution. This change brings backquote substitution in line with the system() function.

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`"
```

Substitution of string variables as macros

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:

```
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
```

The function exists() may be useful in connection with macro evaluation. The following example checks that C can safely be expanded as the name of a user-defined variable:

```
C = "pi"
if (exists(C)) print C," = ", @C
```

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

Macro expansion is handled as the very first thing the interpreter does when looking at a new line of commands and is only done once. Therefore, code like the following will execute correctly:

```
A = "c=1"
@A
```

but this line will not, since the macro is defined on the same line and will not be expanded in time

```
A = "c=1"; @A   # will not expand to c=1
```

Macro expansion inside a bracketed iteration occurs before the loop is executed; i.e. @A will always act as the original value of A even if A itself is reassigned inside the loop.

For execution of complete commands the **evaluate** command may also be handy.

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
```

Syntax

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.

Commas are used to separate coordinates on the **set** commands **arrow**, **key**, and **label**; 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 **set cntrparam** command; the arguments of the **set** commands **dgrid3d**, **dummy**, **isosamples**, **offsets**, **origin**, **samples**, **size**, **time**, and **view**; 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**, **replot** and **splot** commands; and the complete sets of keywords specifying individual plots (data sets or functions) on the **plot**, **replot** and **splot** commands.

Parentheses are used to delimit sets of explicit tics (as opposed to loop parameters) and to indicate computations in the **using** filter of the **fit**, **plot**, **replot** 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 **using** filter of the **plot**, **replot**, **splot** and **fit** commands.

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

Curly braces are used in the syntax for enhanced text mode and to delimit blocks in if/then/else statements. They are also used to denote complex numbers: {3,2} = 3 + 2i.

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 only 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 must 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 (p. 34)**.

Back-quotes are used to enclose system commands for substitution into the command line. See **substitution (p. 57)**.

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 1970. The command **set timefmt** defines the default format for all inputs: data files, ranges, tics, label positions – anything that accepts a time data value defaults to receiving it in this format. Only one default format can be in effect at a given time. Thus if both x and y data in a file are time/date, by default they are interpreted in the same format. However this default can be replaced when reading any particular file or column of input using the **timecolumn** function in the corresponding **using** specifier.

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 **show xrange** will re-interpret the integer according to **timefmt**. If you change **timefmt**, and then **show** the quantity again, it will be displayed in the new **timefmt**. For that matter, if you reset the data type flag for that axis (e.g. **set xdata**), the quantity will be shown in its numerical form.

The commands **set format** or **set tics format** define the format that will be used for tic labels, whether or not input for the specified axis is time/date.

If time/date information is to be plotted from a file, the **using** 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 **time** function can be used to get the current system time. This value can be converted to a date string with the **strftime** function, or it can be used in conjunction with **timecolumn** to generate relative time/date plots. The type of the argument determines what is returned. If the argument is an integer, **time** returns the current time as an integer, in seconds from 1 Jan 1970. If the argument is real (or complex), the result is real as well. The precision of the fractional (sub-second) part depends on your operating system. If the argument is a string, it is assumed to be a format string, and it is passed to **strftime** to provide a formatted time/date string.

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".

Gnuplot tracks time to millisecond precision. Time formats have been modified to match this. Example: print the current time to msec precision

```
print strftime("%H:%M:%.3S %d-%b-%Y",time(0.0))
18:15:04.253 16-Apr-2011
```

See **time_specifiers (p. 151)**.

Part II

Plotting styles

Many plotting styles are available in gnuplot. They are listed alphabetically below. The commands **set style data** and **set style function** change the default plotting style for subsequent **plot** and **splot** commands.

You can also specify the plot style explicitly as part of the **plot** or **splot** command. If you want to mix plot styles within a single plot, you must specify the plot style for each component.

Example:

```
plot 'data' with boxes, sin(x) with lines
```

Each plot style has its own expected set of data entries in a data file. For example, by default the **lines** style expects either a single column of y values (with implicit x ordering) or a pair of columns with x in the first and y in the second. For more information on how to fine-tune how columns in a file are interpreted as plot data, see **using (p. 115)**.

Arrows

The 2D **arrows** style draws an arrow with specified length and orientation angle at each point (x,y). Additional input columns may be used to provide variable (per-datapoint) color information or arrow style. It is identical to the 2D style **with vectors** except that each the arrow head is positioned using length + angle rather than delta_x + delta_y. See **with vectors (p. 80)**.

```
4 columns:  x  y  length  angle
```

The keywords **with arrows** may be followed by inline arrow style properties, a reference to a predefined arrow style, or **arrowstyle variable** to load the index of the desired arrow style for each arrow from a separate column.

length > 0 is interpreted in x-axis coordinates. -1 < **length** < 0 is interpreted in horizontal graph coordinates; i.e. |length| is a fraction of the total graph width. The program will adjust for differences in x and y scaling or plot aspect ratio so that the visual length is independent of the orientation angle.

angle is always specified in degrees.

Bee swarm plots

"Bee swarm" plots result from applying jitter to separate overlapping points. A typical use is to compare the distribution of y values exhibited by two or more categories of points, where the category determines the x coordinate. See the **set jitter (p. 156)** command for how to control the overlap criteria and the displacement pattern used for jittering. The plots in the figure were created by the same plot command but different jitter settings.

```
set jitter
plot $data using 1:2:1 with points lc variable
```

Boxerrorbars

The **boxerrorbars** style is only relevant to 2D data plotting. It is a combination of the **boxes** and **yerrorbars** styles. It requires 3, 4, or 5 columns of data. An additional (4th, 5th or 6th) input column may be used to provide variable (per-datapoint) color information (see **linecolor (p. 49)** and **rgbcolor variable (p. 50)**). The error bar will be drawn in the same color as the border of the box.

```
3 columns:  x  y  ydelta
4 columns:  x  y  ydelta xdelta        # boxwidth != -2
4 columns:  x  y  ylow  yhigh          # boxwidth == -2
5 columns:  x  y  ylow  yhigh  xdelta
```

The boxwidth will come from the fourth column if the y errors are given as "ydelta" and the boxwidth was not previously set 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.

Boxes

In 2D plots the **boxes** style draws a rectangle centered about the given x coordinate that extends from the x axis, i.e. from y=0 not from the graph border, to the given y coordinate. The width of the box can be provided in an additional input column or controlled by **set boxwidth**. Otherwise each box extends to touch the adjacent boxes.

In 3D plots the **boxes** style draws a box centered at the given [x,y] coordinate that extends from the plane at z=0 to the given z coordinate. The width of the box on x can be provided in a separate input column or via **set boxwidth**. The depth of the box on y is controlled by **set boxdepth**. Boxes do not automatically expand to touch each other as in 2D plots.

2D boxes

plot with boxes uses 2 or 3 columns of basic data. Additional input columns may be used to provide information such as variable line or fill color. See **rgbcolor variable (p. 50)**.

```
2 columns:  x  y
3 columns:  x  y  x_width
```

The width of the box is obtained in one of three ways. If the input data has a third column, this will be used to set the box width. Otherwise if a width has been set using the **set boxwidth** command, this will be used. If neither of these is available, the width of each box will be calculated so that it touches the adjacent boxes.

The box interiors are drawn using the current fillstyle. Alternatively a fillstyle may be specified in the plot command. See **set style fill (p. 199)**. If no fillcolor is given in the plot command, the current line color is used.

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 and fillcolors for each dataset:

```
plot 'file1' with boxes fs solid 0.25 fc 'cyan', \
     'file2' with boxes fs solid 0.50 fc 'blue', \
     'file3' with boxes fs solid 0.75 fc 'magenta', \
     'file4' with boxes fill pattern 1, \
     'file5' with boxes fill empty
```

3D boxes

splot with boxes requires at least 3 columns of input data. Additional input columns may be used to provide information such as box width or fill color.

```
3 columns:  x  y  z
4 columns:  x  y  z  [x_width or color]
5 columns:  x  y  z  x_width  color
```

The last column is used as a color only if the splot command specifies a variable color mode. Examples

```
splot 'blue_boxes.dat' using 1:2:3 fc "blue"
splot 'rgb_boxes.dat' using 1:2:3:4 fc rgb variable
splot 'category_boxes.dat' using 1:2:3:4:5 lc variable
```

In the first example all boxes are blue and have the width previously set by **set boxwidth**. In the second example the box width is still taken from **set boxwidth** because the 4th column is interpreted as a 24-bit RGB color. The third example command reads box width from column 4 and interprets the value in column 5 as an integer linetype from which the color is derived.

By default boxes have no thickness; they consist of a single rectangle parallel to the xz plane at the specified y coordinate. You can change this to a true box with four sides and a top by setting a non-zero extent on y. See **set boxdepth (p. 135)**.

3D boxes are processed as pm3d quadrangles rather than as surfaces. Because of this the front/back order of drawing is not affected by **set hidden3d**. Similarly if you want each box face to have a border you must use **set pm3d border** rather than **set style fill border**. See **set pm3d (p. 187)**. For best results use a combination of **set pm3d depthorder base** and **set pm3d lighting**.

Boxplot

Boxplots are a common way to represent a statistical distribution of values. Quartile boundaries are determined such that 1/4 of the points have a value equal or less than the first quartile boundary, 1/2 of the points have a value equal or less than the second quartile (median) value, etc. A box is drawn around the region between the first and third quartiles, with a horizontal line at the median value. Whiskers extend from the box to user-specified limits. Points that lie outside these limits are drawn individually.

Examples

```
# Place a boxplot at x coordinate 1.0 representing the y values in column 5
plot 'data' using (1.0):5

# Same plot but suppress outliers and force the width of the boxplot to 0.3
set style boxplot nooutliers
plot 'data' using (1.0):5:(0.3)
```

By default only one boxplot is produced that represents all y values from the second column of the using specification. However, an additional (fourth) column can be added to the specification. If present, the values of that column will be interpreted as the discrete levels of a factor variable. As many boxplots will be drawn as there are levels in the factor variable. The separation between these boxplots is 1.0 by default, but it can be changed by **set style boxplot separation**. By default, the value of the factor variable is shown as a tic label below (or above) each boxplot.

Example

```
# Suppose that column 2 of 'data' contains either "control" or "treatment"
# The following example produces two boxplots, one for each level of the
# factor
plot 'data' using (1.0):5:(0):2
```

The default width of the box can be set via **set boxwidth <width>** or may be specified as an optional 3rd column in the **using** clause of the plot command. The first and third columns (x coordinate and width) are normally provided as constants rather than as data columns.

By default the whiskers extend from the ends of the box to the most distant point whose y value lies within 1.5 times the interquartile range. By default outliers are drawn as circles (point type 7). The width of the bars at the end of the whiskers may be controlled using **set bars (p. 147)** or **set errorbars (p. 147)**.

These default properties may be changed using the **set style boxplot** command. See **set style boxplot (p. 197)**, **bars (p. 147)**, **boxwidth (p. 134)**, **fillstyle (p. 199)**, **candlesticks (p. 65)**.

Boxxyerror

The **boxxyerror** plot style is only relevant to 2D data plotting. It is similar to the **xyerrorbars** style except that it draws rectangular areas rather than crosses. It uses either 4 or 6 basic columns of input data. Additional input columns may be used to provide information such as variable line or fill color (see **rgbcolor variable (p. 50)**).

```
4 columns:  x  y  xdelta  ydelta
6 columns:  x  y  xlow  xhigh  ylow  yhigh
```

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.

The 6 column form of the command provides a convenient way to plot rectangles with arbitrary x and y bounds.

An additional (5th or 7th) input column may be used to provide variable (per-datapoint) color information (see **linecolor (p. 49)** and **rgbcolor variable (p. 50)**).

The interior of the boxes is drawn according to the current fillstyle. See **set style fill (p. 199)** and **boxes (p. 62)** for details. Alternatively a new fillstyle may be specified in the plot command.

Candlesticks

The **candlesticks** style can be used for 2D data plotting of financial data or for generating box-and-whisker plots of statistical data. 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.

Five columns of basic data are required:

```
financial data:   date  open  low  high  close
whisker plot:     x  box_min  whisker_min  whisker_high  box_high
```

The width of the rectangle can be controlled by the **set boxwidth** command. For backwards compatibility with earlier gnuplot versions, when the boxwidth parameter has not been set then the width of the candlestick rectangle is taken from **set errorbars <width>**.

Alternatively, an explicit width for each box-and-whiskers grouping may be specified in an optional 6th column of data. The width must be given in the same units as the x coordinate.

An additional (6th, or 7th if the 6th column is used for width data) input column may be used to provide variable (per-datapoint) color information (see **linecolor (p. 49)** and **rgbcolor variable (p. 50)**).

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 these whiskerbars extend the full horizontal width of the candlestick, but you can modify this by specifying a fraction of the full width.

The usual convention for financial data is that the rectangle is empty if (open < close) and solid fill if (close < open). This is the behavior you will get if the current fillstyle is set to "empty". See **fillstyle (p. 199)**. If you set the fillstyle to solid or pattern, then this will be used for all boxes independent of open and close values. See also **set errorbars (p. 147)** and **financebars (p. 69)**. See also the `candlestick`

and `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 errorbars 4.0
set style fill empty
plot 'stat.dat' using 1:3:2:6:5 with candlesticks title 'Quartiles', \
     ''         using 1:4:4:4:4 with candlesticks lt -1 notitle

# Plot with crossbars on the whiskers, crossbars are 50% of full width
plot 'stat.dat' using 1:3:2:6:5 with candlesticks whiskerbars 0.5
```

See **set boxwidth (p. 134)**, **set errorbars (p. 147)**, **set style fill (p. 199)**, and **boxplot (p. 63)**.

Circles

The **circles** style plots a circle with an explicit radius at each data point. The radius is always interpreted in the units of the plot's horizontal axis (x or x2). The scale on y and the aspect ratio of the plot are both ignored. If the radius is not given in a separate column for each point it is taken from **set style circle**. In this case the radius may use graph or screen coordinates.

Many combinations of per-point and previously set properties are possible. For 2D plots these include

```
using x:y
using x:y:radius
using x:y:color
using x:y:radius:color
using x:y:radius:arc_begin:arc_end
using x:y:radius:arc_begin:arc_end:color
```

By default a full circle will be drawn. It is possible to instead plot arc segments by specifying a start and end angle (in degrees) in columns 4 and 5.

A per-circle color may be provided in the last column of the using specifier. In this case the plot command must include a corresponding variable color term such as **lc variable** or **fillcolor rgb variable**.

For 3D plots the using specifier must contain

```
splot DATA using x:y:z:radius:color
```

where the variable color column is options. See **set style circle (p. 202)** and **set style fill (p. 199)**.

Examples:

```
# draws circles whose area is proportional to the value in column 3
set style fill transparent solid 0.2 noborder
plot 'data' using 1:2:(sqrt($3)) with circles, \
     'data' using 1:2 with linespoints

# draws Pac-men instead of circles
plot 'data' using 1:2:(10):(40):(320) with circles

# draw a pie chart with inline data
set xrange [-15:15]
set style fill transparent solid 0.9 noborder
plot '-' using 1:2:3:4:5:6 with circles lc var
0    0    5    0    30    1
0    0    5   30    70    2
0    0    5   70   120    3
0    0    5  120   230    4
0    0    5  230   360    5
e
```

The result is similar to using a **points** plot with variable size points and pointstyle 7, except that the circles will scale with the x axis range. See also **set object circle (p. 177)** and **fillstyle (p. 199)**.

Ellipses

The **ellipses** style plots an ellipse at each data point. This style is only relevant for 2D plotting. Each ellipse is described in terms of its center, major and minor diameters, and the angle between its major diameter and the x axis.

```
2 columns: x y
3 columns: x y major_diam
4 columns: x y major_diam minor_diam
5 columns: x y major_diam minor_diam angle
```

If only two input columns are present, they are taken as the coordinates of the centers, and the ellipses will be drawn with the default extent (see **set style ellipse (p. 202)**). The orientation of the ellipse, which is defined as the angle between the major diameter and the plot's x axis, is taken from the default ellipse style (see **set style ellipse (p. 202)**). If three input columns are provided, the third column is used for both diameters. The orientation angle defaults to zero. If four columns are present, they are interpreted as x, y, major diameter, minor diameter. Note that these are diameters, not radii. An optional 5th column may specify the orientation angle in degrees. The ellipses will also be drawn with their default extent if either of the supplied diameters in the 3-4-5 column form is negative.

In all of the above cases, optional variable color data may be given in an additional last (3th, 4th, 5th or 6th) column. See **colorspec (p. 49)**.

By default, the major diameter is interpreted in the units of the plot's horizontal axis (x or x2) while the minor diameter in that of the vertical (y or y2). If the x and y axis scales are not equal, the major/minor diameter ratio will no longer be correct after rotation. This can be changed with the **units** keyword, however.

There are three alternatives: if **units xy** is included in the plot specification, the axes will be scaled as described above. **units xx** ensures that both diameters are interpreted in units of the x axis, while **units yy** means that both diameters are interpreted in units of the y axis. In the latter two cases the ellipses will have the correct aspect ratio, even if the plot is resized. If **units** is omitted from the plot command, the setting from **set style ellipse** will be used.

Example (draws ellipses, cycling through the available line types):

```
plot 'data' using 1:2:3:4:(0):0 with ellipses
```

See also **set object ellipse (p. 176)**, **set style ellipse (p. 202)** and **fillstyle (p. 199)**.

Dots

The **dots** style plots a tiny dot at each point; this is useful for scatter plots with many points. Either 1 or 2 columns of input data are required in 2D. Three columns are required in 3D.

For some terminals (post, pdf) the size of the dot can be controlled by changing the linewidth.

```
1 column    y         # x is row number
2 columns:  x  y
3 columns:  x  y  z   # 3D only (splot)
```

Filledcurves

The **filledcurves** style is only used for 2D plotting. It has three variants. The first two variants require either a single function or two columns (x,y) of input data, and may be further modified by the options listed below.

Syntax:

```
plot ... with filledcurves [option]
```

where the option can be one of the following

```
[closed | {above | below}
{x1 | x2 | y | r}[=<a>] | xy=<x>,<y>]
```

The first variant, **closed**, treats the curve itself as a closed polygon. This is the default if there are two columns of input data.

The second variant is to fill the area between the curve and a given axis, a horizontal or vertical line, or a point.

```
filledcurves closed   ... just filled closed curve,
filledcurves x1       ... x1 axis,
filledcurves x2       ... x2 axis, etc for y1 and y2 axes,
filledcurves y=42     ... line at y=42, i.e. parallel to x axis,
filledcurves xy=10,20 ... point 10,20 of x1,y1 axes (arc-like shape).
filledcurves above r=1.5  the area of a polar plot outside radius 1.5
```

The third variant fills the area between two curves sampled at the same set of x coordinates. It requires three columns of input data (x, y1, y2). This is the default if there are three or more columns of input data. If you have a y value in column 2 and an associated error value in column 3 the area of uncertainty can be represented by shading. See also the similar 3D plot style **zerrorfill (p. 85)**.

```
3 columns:  x  y  yerror

plot $DAT using 1:($2-$3):($2+$3) with filledcurves, \
     $DAT using 1:2 smooth mcs with lines
```

The **above** and **below** options apply both to commands of the form

```
... filledcurves above {x1|x2|y|r}=<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.

Notes: Not all terminal types support this plotting mode.

```
   The x= and y= keywords are ignored for 3 columns data plots
```

Zooming a filled curve drawn from a datafile may produce empty or incorrect areas because gnuplot is clipping points and lines, and not areas.

If the values <x>, <y>, or <a> are outside the drawing boundary they are moved to the graph boundary. Then the actual fill area in the case of option xy=<x>,<y> will depend on xrange and yrange.

Fill properties

Plotting **with filledcurves** can be further customized by giving a fillstyle (solid/transparent/pattern) or a fillcolor. If no fillstyle (**fs**) is given in the plot command then the current default fill style is used. See **set style fill (p. 199)**. If no fillcolor (**fc**) is given in the plot command, the usual linetype color sequence is followed.

The {{no}border} property of the fillstyle is honored by filledcurves mode **closed**, the default. It is ignored by all other filledcurves modes. Example:

```
 plot 'data' with filledcurves fc "cyan" fs solid 0.5 border lc "blue"
```

Financebars

The **financebars** style is only relevant for 2D data plotting of financial data. It requires 1 x coordinate (usually a date) and 4 y values (prices).

```
5 columns:   date  open  low  high  close
```

An additional (6th) input column may be used to provide variable (per-record) color information (see **linecolor (p. 49)** and **rgbcolor variable (p. 50)**).

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 **set errorbars**. The symbol will be unchanged if the high and low prices are interchanged. See **set errorbars (p. 147)** and **candlesticks (p. 65)**, and also the finance demo.

Fsteps

The **fsteps** style is only relevant to 2D 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). The input column requires are the same as for plot styles **lines** and **points**. The difference between **fsteps** and **steps** is that **fsteps** traces first the change in y and then the change in x. **steps** traces first the change in x and then the change in y.

See also `steps demo.`

Fillsteps

The **fillsteps** style is exactly like **steps** except that the area between the curve and y=0 is filled in the current fill style. See **steps (p. 80)**.

Histeps

The **histeps** style is only relevant to 2D 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). The input column requires are the same as for plot styles **lines** and **points**.

If **autoscale** 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.`

Histograms

The **histograms** style is only relevant to 2D plotting. It produces a bar chart from a sequence of parallel data columns. 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
set style histogram {title font "name,size" tc <colorspec>}
```

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 (p. 199)**).

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 **using**.

The **errorbars** style is very similar to the **clustered** style, except that it requires additional columns of input for each entry. The first column holds the height (y value) of that box, exactly as for the **clustered** style.

```
2 columns:        y yerr          bar extends from y-yerr to y+err
3 columns:        y ymin ymax     bar extends from ymin to ymax
```

The appearance of the error bars is controlled by the current value of **set errorbars** 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. Positive values stack upwards from y=0; negative values stack downwards. Mixed positive and negative values will produce both an upward stack and a downward stack. 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 (p. 199)**).

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 **set boxwidth** 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: Suppose that the input file contains data values in columns 2, 4, 6, ... and error estimates in columns 3, 5, 7, ... This example plots the values in columns 2 and 4 as a histogram of clustered boxes (the default style). Because we use iteration in the plot command, any number of data columns can be handled in a single command. See **plot for (p. 122)**.

```
set boxwidth 0.9 relative
set style data histograms
set style histogram cluster
set style fill solid 1.0 border lt -1
plot for [COL=2:4:2] 'file.dat' using COL
```

This will produce a plot with clusters of two 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 for [COL=2:4:2] 'file.dat' using COL:xticlabels(1)
```

If the file contains both magnitude and range information 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 with linewidth 2, using the border linetype from the current fill style.

```
set errorbars fullwidth
set style fill solid 1 border lt -1
set style histogram errorbars gap 2 lw 2
plot for [COL=2:4:2] 'file.dat' using COL:COL+1
```

This shows how to plot the same data as a rowstacked histogram. Just to be different, this example lists the separate columns explicitly rather than using iteration.

```
set style histogram rowstacked
plot 'file.dat' using 2, '' using 4:xtic(1)
```

This will produce a plot in which each vertical bar corresponds to one row of data. Each vertical bar contains a stack of two segments, corresponding in height to the values found in columns 2 and 4 of the datafile.

Finally, the commands

```
set style histogram columnstacked
plot 'file.dat' using 2, '' using 4
```

will produce two vertical stacks, one for each column of data. 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.

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 accordingly. See the comments given below.

```
set style histogram columnstacked
plot '' u 5:key(1)            # uses first column to generate key titles
plot '' u 5 title columnhead  # uses first row to generate xtic labels
```

Note that the two examples just given present exactly the same data values, but in different formats.

Newhistogram

Syntax:

```
newhistogram {"<title>" {font "name,size"} {tc <colorspec>}}
             {lt <linetype>} {fs <fillstyle>} {at <x-coord>}
```

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 **newhistogram** command. For example

```
set style histogram  cluster
plot newhistogram "Set A", 'a' using 1, '' using 2, '' using 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, '' using 3, \
     newhistogram "Set B" lt 4, 'b' using 1, '' using 2, '' using 3
```

Similarly you can force the next histogram to begin with a specified fillstyle. If the fillstyle is set to **pattern**, then the pattern used for filling will be incremented automatically.

The **at <x-coord>** option sets the x coordinate position of the following histogram to <x-coord>. For example

```
set style histogram cluster
set style data histogram
set style fill solid 1.0 border -1
set xtic 1 offset character 0,0.3
plot newhistogram "Set A", \
     'file.dat' u 1 t 1, '' u 2 t 2, \
     newhistogram "Set B" at 8, \
     'file.dat' u 2 t 2, '' u 2 t 2
```

will position the second histogram to start at x=8.

Automated iteration over multiple columns

If you want to create a histogram from many columns of data in a single file, it is very convenient to use the plot iteration feature. See **plot for (p. 122)**. For example, to create stacked histograms of the data in columns 3 through 8

```
set style histogram columnstacked
plot for [i=3:8] "datafile" using i title columnhead
```

Image

The **image**, **rgbimage**, and **rgbalpha** plotting styles all project a uniformly sampled grid of data values onto a plane in either 2D or 3D. The input data may be an actual bitmapped image, perhaps converted from a standard format such as PNG, or a simple array of numerical values.

This figure illustrates generation of a heat map from an array of scalar values. The current palette is used to map each value onto the color assigned to the corresponding pixel.

```
plot '-' matrix with image
5 4 3 1 0
2 2 0 0 1
0 0 0 1 0
0 1 2 4 3
e
e
```

Each pixel (data point) of the input 2D image will become a rectangle or parallelipiped in the plot. The coordinates of each data point will determine the center of the parallelipiped. That is, an M x N set of data will form an image with M x N pixels. This is different from the pm3d plotting style, where an M x N set of data will form a surface of (M-1) x (N-1) elements. The scan directions for a binary image data grid can be further controlled by additional keywords. See **binary keywords flipx (p. 105)**, **keywords center (p. 106)**, and **keywords rotate (p. 106)**.

Image data can be scaled to fill a particular rectangle within a 2D plot coordinate system by specifying the x and y extent of each pixel. See **binary keywords dx (p. 105)** and **dy (p. 105)**. To generate the figure at the right, the same input image was placed multiple times, each with a specified dx, dy, and origin. The input PNG image of a building is 50x128 pixels. The tall building was drawn by mapping this using **dx=0.5 dy=1.5**. The short building used a mapping **dx=0.5 dy=0.35**.

The **image** style handles input pixels containing a grayscale or color palette value. Thus 2D plots (**plot** command) require 3 columns of data (x,y,value), while 3D plots (**splot** command) require 4 columns of data (x,y,z,value).

The **rgbimage** style handles input pixels that are described by three separate values for the red, green, and blue components. Thus 5D data (x,y,r,g,b) is needed for **plot** and 6D data (x,y,z,r,g,b) for **splot**. The individual red, green, and blue components are assumed to lie in the range [0:255]. This matches the convention used in PNG and JPEG files (see **binary filetype (p. 104)**). However some data files use an alternative convention in which RGB components are floating point values in the range [0:1]. To use the **rgbimage** style with such data, first use the command **set rgbmax 1.0**.

The **rgbalpha** style handles input pixels that contain alpha channel (transparency) information in addition to the red, green, and blue components. Thus 6D data (x,y,r,g,b,a) is needed for **plot** and 7D data (x,y,z,r,g,b,a) for **splot**. The r, g, b, and alpha components are assumed to lie in the range [0:255]. To plot data for which RGBA components are floating point values in the range [0:1], first use the command **set rgbmax 1.0**.

If only a single data column is provided for the color components of either rgbimage or rgbalpha plots, it is interpreted as containing 32 bit packed ARGB data using the convention that alpha=0 means opaque and alpha=255 means fully transparent. Note that this is backwards from the alpha convention if alpha is supplied in a separate column, but matches the ARGB packing convention for individual commands to set color. See **colorspec (p. 49)**.

Transparency

The **rgbalpha** plotting style assumes that each pixel of input data contains an alpha value in the range [0:255]. A pixel with alpha = 0 is purely transparent and does not alter the underlying contents of the plot. A pixel with alpha = 255 is purely opaque. All terminal types can handle these two extreme cases. A pixel with 0 < alpha < 255 is partially transparent. Terminal types that do not support partial transparency will round this value to 0 or 255.

Image pixels

Some terminals use device- or library-specific optimizations to render image data within a rectangular 2D area. This sometimes produces undesirable output, e.g. bad clipping or scaling, missing edges. The **pixels** keyword tells gnuplot to use generic code that renders the image pixel-by-pixel instead. This rendering mode is slower and may result in much larger output files, but should produce a consistent rendered view on all terminals. Example:

```
plot 'data' with image pixels
```

Impulses

The **impulses** style displays a vertical line from y=0 to the y value of each point (2D) or from z=0 to the z value of each point (3D). Note that the y or z values may be negative. Data from additional columns can be used to control the color of each impulse. To use this style effectively in 3D plots, it is useful to choose thick lines (linewidth > 1). This approximates a 3D bar chart.

```
1 column:   y
2 columns:  x  y     # line from [x,0] to [x,y]  (2D)
3 columns:  x  y  z  # line from [x,y,0] to [x,y,z] (3D)
```

Labels

The **labels** style reads coordinates and text from a data file and places the text string at the corresponding 2D or 3D position. 3 or 4 input columns of basic data are required. Additional input columns may be used to provide properties that vary point by point such as text rotation angle (keywords **rotate variable**) or color (see **textcolor variable (p. 50)**).

```
3 columns:  x  y  string    # 2D version
4 columns:  x  y  z  string # 3D version
```

The font, color, rotation angle and other properties of the printed text may be specified as additional command options (see **set label (p. 161)**). The example below generates a 2D plot with text labels constructed from the city whose name is taken from column 1 of the input file, and whose geographic coordinates are in columns 4 and 5. The font size is calculated from the value in column 3, in this case the population.

```
CityName(String,Size) = sprintf("{/=%d %s}", Scale(Size), String)
plot 'cities.dat' using 5:4:(CityName(stringcolumn(1),$3)) with labels
```

If we did not want to adjust the font size to a different size for each city name, the command would be much simpler:

```
plot 'cities.dat' using 5:4:1 with labels font "Times,8"
```

If the labels are marked as **hypertext** then the text only appears if the mouse is hovering over the corresponding anchor point. See **hypertext (p. 164)**. In this case you must enable the label's **point** attribute so that there is a point to act as the hypertext anchor:

```
plot 'cities.dat' using 5:4:1 with labels hypertext point pt 7
```

The **labels** style can also be used in place of the **points** style when the set of predefined point symbols is not suitable or not sufficiently flexible. For example, here we define a set of chosen single-character symbols and assign one of them to each point in a plot based on the value in data column 3:

```
set encoding utf8
symbol(z) = "•□+⊙♠♣♡◇"[int(z):int(z)]
splot 'file' using 1:2:(symbol($3)) with labels
```

This example shows use of labels with variable rotation angle in column 4 and textcolor ("tc") in column 5. Note that variable color is always taken from the last column in the **using** specifier.

```
plot $Data using 1:2:3:4:5 with labels tc variable rotate variable
```

Lines

The **lines** style connects adjacent points with straight line segments. It may be used in either 2D or 3D plots. The basic form requires 1, 2, or 3 columns of input data. Additional input columns may be used to provide information such as variable line color (see **rgbcolor variable (p. 50)**).

2D form (no "using" spec)

```
1 column:   y       # implicit x from row number
2 columns:  x  y
```

3D form (no "using" spec)

```
1 column:   z       # implicit x from row, y from index
3 columns:  x  y  z
```

See also **linetype (p. 164)**, **linewidth (p. 200)**, and **linestyle (p. 200)**.

Linespoints

The **linespoints** style (short form **lp**) connects adjacent points with straight line segments and then goes back to draw a small symbol at each point. Points are drawn with the default size determined by **set pointsize** unless a specific point size is given in the plot command or a variable point size is provided in an additional column of input data. Additional input columns may also be used to provide information such as variable line color. See **lines (p. 76)** and **points (p. 78)**.

Two keywords control whether or not every point in the plot is marked with a symbol, **pointinterval** (short form **pi**) and **pointnumber** (short form **pn**).

pi N or **pi -N** tells gnuplot to only place a symbol on every Nth point. A negative value for N will erase the portion of line segment that passes underneath the symbol. The size of the erased portion is controlled by **set pointintervalbox**.

pn N or **pn -N** tells gnuplot to label only N of the data points, evenly spaced over the data set. As with **pi**, a negative value for N will erase the portion of line segment that passes underneath the symbol.

Parallelaxes

Parallel axis plots can highlight correlation in a multidimensional data set. Individual columns of input data are each associated with a separately scaled vertical axis. If all columns are drawn from a single file then each line on the plot represents values from a single row of data in that file. It is common to use some discrete categorization to assign line colors, allowing visual exploration of the correlation between this categorization and the axis dimensions.

Syntax:

```
set style data parallelaxes
plot $DATA using col1{:varcol1} {at <xpos>} {<line properties}, \
     $DATA using col2, ...
```

CHANGE: Version 5.4 of gnuplot introduces a change in the syntax for plot style parallelaxes. The revised syntax allows an unlimited number of parallel axes.

```
gnuplot 5.2:   plot $DATA using 1:2:3:4:5 with parallelaxes
gnuplot 5.4:   plot for [col=1:5] $DATA using col with parallelaxes
```

The new syntax also allows explicit placement of the parallel vertical axes along the x axis as in the example below. If no explicit x coordinate is provide axis N will be placed at x=N.

```
array xpos[5] = [1, 5, 6, 7, 11, 12]
plot for [col=1:5] $DATA using col with parallelaxes at xpos[col]
```

By default gnuplot will automatically determine the range and scale of the individual axes from the input data, but the usual **set axis range** commands can be used to customize this. See **set paxis (p. 186)**.

Polar plots

Polar plots are generated by changing the current coordinate system to polar before issuing a plot command. The option **set polar** tells gnuplot to interpret input 2D coordinates as <angle>,<radius> rather than <x>,<y>. Many, but not all, of the 2D plotting styles work in polar mode. The figure shows a combination of plot styles **lines** and **filledcurves**. See **set polar (p. 192)**, **set rrange (p. 194)**, **set size square (p. 195)**, **set theta (p. 206)**, **set ttics (p. 209)**.

Points

The **points** style displays a small symbol at each point. The command **set pointsize** may be used to change the default size of all points. The point type defaults to that of the linetype. See **linetype (p. 164)**. If no **using** spec is found in the plot command, input data columns are interpreted implicitly in the order

```
x y pointsize pointtype color
```

Any columns beyond the first two (x and y) are optional; they correspond to additional plot properties **pointsize variable**, **pointtype variable**, etc.

The first 8 point types are shared by all terminals. Individual terminals may provide a much larger number of distinct point types. Use the **test** command to show what is provided by the current terminal settings.

Alternatively any single printable character may be given instead of a numerical point type, as in the example below. You may use any unicode character as the pointtype (assumes utf8 support). See **escape sequences (p. 35)**. Longer strings may be plotted using plot style **labels** rather than **points**.

```
plot f(x) with points pt "#"
plot d(x) with points pt "\U+2299"
```

When using the keywords **pointtype**, **pointsize**, or **linecolor** in a plot command, the additional keyword **variable** may be given instead of a number. In this case the corresponding properties of each point are assigned by additional columns of input data. Variable pointsize is always taken from the first additional column provided in a **using** spec. Variable color is always taken from the last additional column. See **colorspec (p. 49)**. If all three properties are specified for each point, the order of input data columns is thus

```
plot DATA using x:y:pointsize:pointtype:color \
     with points lc variable pt variable ps variable
```

Note: for information on user-defined program variables, see **variables (p. 43)**.

Polygons

```
splot DATA {using x:y:z} with polygons
      {fillstyle <fillstyle spec>}
      {fillcolor <colorspec>}
```

splot with polygons uses pm3d to render individual triangles, quadrangles, and larger polygons in 3D. These may be facets of a 3D surface or isolated shapes. The code assumes that the vertices lie in a plane. Vertices defining individual polygons are read from successive records of the input file. A blank line separates one polygon from the next.

The fill style and color may be specified in the splot command, otherwise the global fillstyle from **set style fill** is used. Due to limitations in the pm3d code, a single border line style from **set pm3d border** is applied to all polygons. This restriction may be removed in a later gnuplot version.

pm3d sort order and lighting are applied to the faces. It is probably always desirable to use **set pm3d depthsort**.

```
set xyplane at 0
set view equal xyz
unset border
unset tics
set pm3d depth
set pm3d border lc "black" lw 1.5
splot 'icosahedron.dat' with polygons \
      fs transparent solid 0.8 fc bgnd
```

Spiderplot

Spider plots are essentially parallel axis plots in which the axes are arranged radially rather than vertically. Such plots are sometimes called **rader charts**. In gnuplot this requires working within a coordinate system established by the command **set spiderplot**, analogous to **set polar** except that the angular coordinate is determined implicitly by the parallel axis number. The appearance, labelling, and tic placement of the axes is controlled by **set paxis**. Further style choices are controlled using **set style spiderplot (p. 203)**, **set grid (p. 153)**, and the individual components of the plot command.

Because each spider plot corresponds to a row of data rather than a column, it would make no sense to generate key entry titles in the normal way. Instead, if a plot component contains a title the text is used to label the corresponding axis. This overrides any previous **set paxis n label "Foo"**. To place a title in the key, you can either use a separate **keyentry** command or extract text from a column in the input file with the **key(column)** using specifier. See **keyentry (p. 159)**, **using key (p. 117)**.

In this figure a spiderplot with 5 axes is used to compare multiple entities that are each characterized by five scores. Each line (row) in $DATA generates a new polygon on the plot.

```
set spiderplot
set style spiderplot fs transparent solid 0.2 border
set for [p=1:5] paxis p range [0:100]
set for [p=2:5] paxis p tics format ""
set             paxis 1 tics font ",9"
set for [p=1:5] paxis p label sprintf("Score %d",p)
set grid spiderplot
plot for [i=1:5] $DATA using i:key(1)
```

Newspiderplot

Normally the sequential elements of a plot command **with spiderplot** each correspond to one vertex of a single polygon. In order to describe multiple polygons in the same plot command, they must be separated by **newspiderplot**. Example:

```
# One polygon with 10 vertices
plot for [i=1:5] 'A' using i, for [j=1:5] 'B' using j
# Two polygons with 5 vertices
plot for [i=1:5] 'A' using i, newspiderplot, for [j=1:5] 'B' using j
```

Steps

The **steps** style is only relevant to 2D 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). The input column requires are the same as for plot styles **lines** and **points**. The difference between **fsteps** and **steps** is that **fsteps** traces first the change in y and then the change in x. **steps** traces first the change in x and then the change in y. To fill the area between the curve and the baseline at y=0, use **fillsteps**. See also `steps demo.`

Rgbalpha

See **image (p. 73)**.

Rgbimage

See **image (p. 73)**.

Vectors

The 2D **vectors** style draws a vector from (x,y) to (x+xdelta,y+ydelta). The 3D **vectors** style is similar, but requires six columns of basic data. In both cases, an additional input column (5th in 2D, 7th in 3D) may be used to provide variable (per-datapoint) color information. (see **linecolor (p. 49)** and **rgbcolor variable (p. 50)**). A small arrowhead is drawn at the end of each vector.

```
4 columns:  x  y  xdelta  ydelta
6 columns:  x  y  z  xdelta  ydelta  zdelta
```

The keywords "with vectors" may be followed by an inline arrow style specifications, a reference to a predefined arrow style, or a request to read the index of the desired arrow style for each vector from a separate column. Note: If you choose "arrowstyle variable" it will fill in all arrow properties at the time the corresponding vector is drawn; you cannot mix this keyword with other line or arrow style qualifiers in the plot command.

```
plot ... with vectors filled heads
plot ... with vectors arrowstyle 3
plot ... using 1:2:3:4:5 with vectors arrowstyle variable
```

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 vectors filled head lw 2
```

splot with vectors is supported only for **set mapping cartesian**. **set clip one** and **set clip two** affect vectors drawn in 2D. See **set clip (p. 135)** and **arrowstyle (p. 196)**.

See also the 2D plot style **with arrows (p. 61)** that is identical to **with vectors (p. 80)** except that each arrow is specified using x:y:length:angle.

Xerrorbars

The **xerrorbars** style is only relevant to 2D data plots. **xerrorbars** is like **points**, 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. The appearance of the tic mark at the ends of the bar is controlled by **set errorbars**. The basic style requires either 3 or 4 columns:

```
3 columns:  x  y  xdelta
4 columns:  x  y  xlow  xhigh
```

An additional input column (4th or 5th) may be used to provide information such as variable point color.

Xyerrorbars

The **xyerrorbars** style is only relevant to 2D data plots. **xyerrorbars** is like **points**, 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. The appearance of the tic mark at the ends of the bar is controlled by **set errorbars**. Either 4 or 6 input columns are required.

```
4 columns:  x  y  xdelta  ydelta
6 columns:  x  y  xlow  xhigh  ylow  yhigh
```

If data are provided in an unsupported mixed form, the **using** 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
```

An additional input column (5th or 7th) may be used to provide variable (per-datapoint) color information.

Yerrorbars

The **yerrorbars** (or **errorbars**) style is only relevant to 2D 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. The appearance of the tic mark at the ends of the bar is controlled by **set errorbars**.

```
2 columns:  [implicit x] y ydelta
3 columns:  x  y  ydelta
4 columns:  x  y  ylow  yhigh
```

An additional input column (4th or 5th) may be used to provide information such as variable point color.

See also errorbar demo.

Xerrorlines

The **xerrorlines** style is only relevant to 2D 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. The appearance of the tic mark at the ends of the bar is controlled by **set errorbars**. The basic style requires either 3 or 4 columns:

```
3 columns:  x  y  xdelta
4 columns:  x  y  xlow  xhigh
```

An additional input column (4th or 5th) may be used to provide information such as variable point color.

Xyerrorlines

The **xyerrorlines** style is only relevant to 2D 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. The appearance of the tic mark at the ends of the bar is controlled by **set errorbars**. Either 4 or 6 input columns are required.

```
4 columns:  x  y  xdelta  ydelta
6 columns:  x  y  xlow  xhigh  ylow  yhigh
```

If data are provided in an unsupported mixed form, the **using** 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
```

An additional input column (5th or 7th) may be used to provide variable (per-datapoint) color information.

Yerrorlines

The **yerrorlines** (or **errorlines**) style is only relevant to 2D 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. The appearance of the tic mark at the ends of the bar is controlled by **set errorbars**. Either 3 or 4 input columns are required.

```
3 columns:  x  y  ydelta
4 columns:  x  y  ylow  yhigh
```

An additional input column (4th or 5th) may be used to provide information such as variable point color.

See also `errorbar demo.`

3D plots

3D plots are generated using the command **splot** rather than **plot**. Many of the 2D plot styles (points, images, impulse, labels, vectors) can also be used in 3D by providing an extra column of data containing z coordinate. Some plot types (pm3d coloring, surfaces, contours) must be generated using the **splot** command even if only a 2D projection is wanted.

Surface plots

The styles **splot with lines** and **splot with surface** both generate a surface made from a grid of lines. Solid surfaces can be generated using the style **splot with pm3d**. Usually the surface is displayed at some convenient viewing angle, such that it clearly represents a 3D surface. See **set view (p. 211)**. In this case the X, Y, and Z axes are all visible in the plot. The illusion of 3D is enhanced by choosing hidden line removal. See **hidden3d (p. 154)**. The **splot** command can also calculate and draw contour lines corresponding to constant Z values. These contour lines may be drawn onto the surface itself, or projected onto the XY plane. See **set contour (p. 140)**.

2D projection (set view map)

An important special case of the **splot** command is to map the Z coordinate onto a 2D surface by projecting the plot along the Z axis onto the xy plane. See **set view map (p. 211)**. This plot mode is useful for contour plots and heat maps. This figure shows contours plotted once with plot style **lines** and once with style **labels**.

PM3D plots

3D surfaces can also be drawn using solid pm3d quadrangles rather than lines. In this case there is no hidden surface removal, but if the component facets are drawn back-to-front then a similar effect is achieved. See **set pm3d depthorder (p. 189)**. While pm3d surfaces are by default colored using a smooth color palette (see **set palette (p. 180)**), it is also possible to specify a solid color surface or to specify distinct solid colors for the top and bottom surfaces as in the figure shown here. See **pm3d fillcolor (p. 191)**. Unlike the line-trimming in hidden3d mode, pm3d surfaces can be smoothly clipped to the current zrange. See **set pm3d clipping (p. 190)**.

Fence plots

Fence plots combine several 2D plots by aligning their Y coordinates and separating them from each other by a displacement along X. Filling the area between a base value and each plot's series of Z values enhances the visual impact of the alignment on Y and comparison on Z. There are several ways such plots can be created in gnuplot. The simplest is to use the 5 column variant of the **zerrorfill** style. Suppose there are separate curves z = Fi(y) indexed by i. A fence plot is generated by **splot with zerrorfill** using input columns

```
i y z_base z_base Fi(y)
```

Isosurface

This 3D plot style requires a populated voxel grid (see **set vgrid (p. 210)**, **vfill (p. 235)**). Linear interpolation of voxel grid values is used to estimate fractional grid coordinates corresponding to the requested isolevel. These points are then used to generate a tessellated surface. The facets making up the surface are rendered as pm3d polygons, so the surface coloring, transparency, and border properties are controlled by

set pm3d. In general the surface is easier to interpret visually if facets are given a thin border that is darker than the fill color. By default the tessellation uses a mixture of quadrangles and triangles. To use triangle only, see **set isosurface (p. 156)**. Example:

```
set style fill solid 0.3
set pm3d depthorder border lc "blue" lw 0.2
splot $helix with isosurface level 10 fc "cyan"
```

Zerrorfill

Syntax:

```
splot DATA using 1:2:3:4[:5] with zerrorfill {fc|fillcolor <colorspec>}
           {lt|linetype <n>} {<line properties>}
```

The **zerrorfill** plot style is similar to one variant of the 2D plot style **filledcurves**. It fills the area between two functions or data lines that are sampled at the same x and y points. It requires 4 or 5 input columns:

```
4 columns:  x  y  z  zdelta
5 columns:  x  y  z  zlow  zhigh
```

The area between zlow and zhigh is filled and then a line is drawn through the z values. By default both the line and the fill area use the same color, but you can change this in the splot command. The fill area properties are also affected by the global fill style; see **set style fill (p. 199)**.

If there are multiple curves in the splot command each new curve may occlude all previous curves. To get proper depth sorting so that curves can only be occluded by curves closer to the viewer, use **set pm3d depthorder base**. Unfortunately this causes all the filled areas to be drawn after all of the corresponding lines of z values. In order to see both the lines and the depth-sorted fill areas you probably will need to make the fill areas partially transparent or use pattern fill rather than solid fill.

The fill area in the first two examples below is the same.

```
splot 'data' using 1:2:3:4 with zerrorfill fillcolor "grey" lt black
splot 'data' using 1:2:3:($3-$4):($3+$4) with zerrorfill
splot '+' using 1:(const):(func1($1)):(func2($1)) with zerrorfill
splot for [k=1:5] datafile[k] with zerrorfill lt black fc lt (k+1)
```

This plot style can also be used to create fence plots. See **fenceplots (p. 84)**.

Part III
Commands

This section lists the commands acceptable to **gnuplot** in alphabetical order. Printed versions of this document contain all commands; the text available interactively 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.

Break

The **break** command is only meaningful inside the bracketed iteration clause of a **do** or **while** statement. It causes the remaining statements inside the bracketed clause to be skipped and iteration is terminated. Execution resumes at the statement following the closing bracket. See also **continue (p. 88)**.

Cd

The **cd** command changes the working directory.

Syntax:

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

The directory name must be enclosed in quotes.

Examples:

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

It is recommended that Windows users use single-quotes, because 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"
```

work as expected.

Call

The **call** command is identical to the **load** command with one exception: the name of the file being loaded may be followed by up to nine parameters.

```
call "inputfile" <param-1> <param-2> <param-3> ... <param-9>
```

Previous versions of gnuplot performed macro-like substitution of the special tokens $0, $1, ... $9 with the literal contents of these parameters. This mechanism is now deprecated (see **call old-style (p. 88)**).

Gnuplot now provides a set of string variables ARG0, ARG1, ..., ARG9 and an integer variable ARGC. When a **call** command is executed ARG0 is set to the name of the input file, ARGC is set to the number of parameters present, and ARG1 to ARG9 are loaded from the parameters that follow it on the command line. Any existing contents of the ARG variables are saved and restored across a **call** command.

Because the parameters ARG1 ... ARG9 are stored in ordinary string variables they may be dereferenced by macro expansion (analogous to the older deprecated syntax). However in many cases it is more natural to use them as you would any other variable.

In parallel to the string parameters ARG1 ... ARG9, the passed parameters are stored in an array ARGV[9]. See **argv (p. 87)**.

Argv[ ]

When a gnuplot script is entered via the **call** command any parameters passed by the caller are available via two mechanisms. Each parameter is stored as a string in variables ARG1, ARG2, ... ARG9. Each parameter is also stored as one element of the array ARGV[9]. Numerical values are stored as complex variables. All other values are stored as strings. Thus after a call

```
call 'routine_1.gp'  1 pi "title"
```

The three arguments are available inside routine_1.gp as follows

```
ARG1 = "1"          ARGV[1] = 1.0
ARG2 = "3.14159"    ARGV[2] = 3.14159265358979...
ARG3 = "title"      ARGV[3] = "title"
```

In this example ARGV[1] and ARGV[2] have the full precision of a floating point variable. ARG2 lost precision in being stored as a string using format "%g".

Example

```
Call site
    MYFILE = "script1.gp"
    FUNC = "sin(x)"
    call MYFILE FUNC 1.23 "This is a plot title"
Upon entry to the called script
    ARG0 holds "script1.gp"
    ARG1 holds the string "sin(x)"
    ARG2 holds the string "1.23"
    ARG3 holds the string "This is a plot title"
    ARGC is 3
The script itself can now execute
    plot @ARG1 with lines title ARG3
    print ARG2 * 4.56, @ARG2 * 4.56
    print "This plot produced by script ", ARG0
```

Notice that because ARG1 is a string it must be dereferenced as a macro, but ARG2 may be dereferenced either as a macro (yielding a numerical constant) or a variable (yielding that same numerical value after auto-promotion of the string "1.23" to a real).

The same result could be obtained directly from a shell script by invoking gnuplot with the **-c** command line option:

```
gnuplot -persist -c "script1.gp" "sin(x)" 1.23 "This is a plot title"
```

Old-style

This describes the deprecated call mechanism used by old versions of gnuplot.

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

The name of the input file must be enclosed in quotes. As each line is read from the input file, it is scanned for the following special character sequences: $0 $1 $2 $3 $4 $5 $6 $7 $8 $9 $#. If found, the sequence **$**+digit is replaced by the corresponding parameter from the **call** command line. Quote characters are not copied and string variable substitution is not performed. The character sequence **$#** is replaced by the number of passed parameters. **$** followed by any other character is treated as an escape sequence; use **$$** to get a single **$**.

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
```

NOTES: This use of the **$** character conflicts both with gnuplot's own syntax for datafile columns and with the use of **$** to indicate environmental variables in a unix-like shell. The special sequence **$#** was mis-interpreted as a comment delimiter in gnuplot versions 4.5 through 4.6.3. Quote characters are ignored during substitution, so string constants are easily corrupted.

Clear

The **clear** command erases the current screen or output device as specified by **set terminal** and **set output**. This usually generates a formfeed on hardcopy devices.

For some terminals **clear** erases only the portion of the plotting surface defined by **set size**, so for these it can be used in conjunction with **set multiplot** 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 **set multiplot (p. 171)**, **set size (p. 195)**, and **set origin (p. 178)** for details.

Continue

The **continue** command is only meaningful inside the bracketed iteration clause of a **do** or **while** statement. It causes the remaining statements inside the bracketed clause to be skipped. Execution resumes at the start of the next iteration (if any remain in the loop condition). See also **break (p. 86)**.

Do

Syntax:

```
do for <iteration-spec> {
     <commands>
     <commands>
}
```

Execute a sequence of commands multiple times. The commands must be enclosed in curly brackets, and the opening "{" must be on the same line as the **do** keyword. This command cannot be used with old-style (un-bracketed) if/else statements. See **if (p. 98)**. For examples of iteration specifiers, see **iteration (p. 47)**. Example:

```
set multiplot layout 2,2
do for [name in "A B C D"] {
    filename = name . ".dat"
    set title sprintf("Condition %s",name)
    plot filename title name
}
unset multiplot
```

See also **while (p. 235)**, **continue (p. 88)**, **break (p. 86)**.

Evaluate

The **evaluate** command executes the commands given as an argument string. Newline characters are not allowed within the string.

Syntax:

```
eval <string expression>
```

This is especially useful for a repetition of similar commands.

Example:

```
set_label(x, y, text) \
  = sprintf("set label '%s' at %f, %f point pt 5", text, x, y)
eval set_label(1., 1., 'one/one')
eval set_label(2., 1., 'two/one')
eval set_label(1., 2., 'one/two')
```

Please see **substitution macros (p. 57)** for another way to execute commands from a string.

Exit

```
exit
exit message "error message text"
exit status <integer error code>
```

The commands **exit** and **quit**, as well as the END-OF-FILE character (usually Ctrl-D) terminate input from the current input stream: terminal session, pipe, or 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"
```

The command **exit error "error message"** simulates a program error. In interactive mode it prints the error message and returns to the command line, breaking out of all nested loops or calls. In non-interactive mode the program will exit.

When gnuplot exits to the controlling shell, the return value is not usually informative. This variant of the command allows you to return a specific value.

```
exit status <value>
```

See help for **batch/interactive (p. 30)** for more details.

Fit

The **fit** command fits a user-supplied real-valued expression to a set of data points, using the nonlinear least-squares Marquardt-Levenberg algorithm. There can be up to 12 independent variables, there is always 1 dependent variable, and any number of parameters can be fitted. Optionally, error estimates can be input for weighting the data points.

The basic use of **fit** is best explained by a simple example:

```
f(x) = a + b*x + c*x**2
fit f(x) 'measured.dat' using 1:2 via a,b,c
plot 'measured.dat' u 1:2, f(x)
```

Syntax:

```
fit {<ranges>} <expression>
    '<datafile>' {datafile-modifiers}
    {{unitweights} | {y|xy|z}error | errors <var1>{,<var2>,...}}
    via '<parameter file>' | <var1>{,<var2>,...}
```

Ranges may be specified to filter the data used in fitting. Out-of-range data points are ignored. The syntax is

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

analogous to **plot**; see **plot ranges (p. 119)**.

<expression> can be any valid **gnuplot** expression, although the most common is a previously user-defined function of the form f(x) or f(x,y). It must be real-valued. The names of the independent variables are set by the **set dummy** command, or in the <ranges> part of the command (see below); by default, the first two are called x and y. Furthermore, the expression should depend on one or more variables whose value is to be determined by the fitting procedure.

<datafile> is treated as in the **plot** command. All the **plot datafile** modifiers (**using**, **every**,...) except **smooth** are applicable to **fit**. See **plot datafile (p. 106)**.

The datafile contents can be interpreted flexibly by providing a **using** qualifier as with plot commands. For example to generate the independent variable x as the sum of columns 2 and 3, while taking z from column 6 and requesting equal weights:

```
fit ... using ($2+$3):6
```

In the absence of a **using** specification, the fit implicitly assumes there is only a single independent variable. If the file itself, or the using specification, contains only a single column of data, the line number is taken as the independent variable. If a **using** specification is given, there can be up to 12 independent variables (and more if specially configured at compile time).

The **unitweights** option, which is the default, causes all data points to be weighted equally. This can be changed by using the **errors** keyword to read error estimates of one or more of the variables from the data file. These error estimates are interpreted as the standard deviation s of the corresponding variable value and used to compute a weight for the datum as 1/s**2.

In case of error estimates of the independent variables, these weights are further multiplied by fitting function derivatives according to the "effective variance method" (Jay Orear, Am. J. Phys., Vol. 50, 1982).

The **errors** keyword is to be followed by a comma-separated list of one or more variable names for which errors are to be input; the dependent variable z must always be among them, while independent variables are optional. For each variable in this list, an additional column will be read from the file, containing that variable's error estimate. Again, flexible interpretation is possible by providing the **using** qualifier. Note that the number of independent variables is thus implicitly given by the total number of columns in the **using** qualifier, minus 1 (for the dependent variable), minus the number of variables in the **errors** qualifier.

As an example, if one has 2 independent variables, and errors for the first independent variable and the dependent variable, one uses the **errors x,z** qualifier, and a **using** qualifier with 5 columns, which are interpreted as x:y:z:sx:sz (where x and y are the independent variables, z the dependent variable, and sx and sz the standard deviations of x and z).

A few shorthands for the **errors** qualifier are available: **yerrors** (for fits with 1 column of independent variable), and **zerrors** (for the general case) are all equivalent to **errors z**, indicating that there is a single extra column with errors of the dependent variable.

xyerrors, for the case of 1 independent variable, indicates that there are two extra columns, with errors of both the independent and the dependent variable. In this case the errors on x and y are treated by Orear's effective variance method.

Note that **yerror** and **xyerror** are similar in both form and interpretation to the **yerrorlines** and **xyerrorlines** 2D plot styles.

With the command **set fit v4** the fit command syntax is compatible with **gnuplot** version 4. In this case there must be two more **using** qualifiers (z and s) than there are independent variables, unless there is only one variable. **gnuplot** then uses the following formats, depending on the number of columns given in the **using** specification:

```
z                           # 1 independent variable (line number)
x:z                         # 1 independent variable (1st column)
x:z:s                       # 1 independent variable (3 columns total)
x:y:z:s                     # 2 independent variables (4 columns total)
x1:x2:x3:z:s                # 3 independent variables (5 columns total)
x1:x2:x3:...:xN:z:s         # N independent variables (N+2 columns total)
```

Please beware that this means that you have to supply z-errors s in a fit with two or more independent variables. If you want unit weights you need to supply them explicitly by using e.g. then format x:y:z:(1).

The dummy variable names may be changed when specifying a range as noted above. The first range corresponds to the first **using** spec, and so on. A range may also be given for z (the dependent variable), in which case data points for which f(x,...) is out of the z range will not contribute to the residual being minimized.

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 **fit multi-branch (p. 96)**.

The **via** qualifier specifies which parameters are to be optimized, 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
set 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 yerror via a, b, c
fit g(x,y) 'surface.dat' using 1:2:3 via a, b, c
fit a0 + a1*x/(1 + a2*x/(1 + a3*x)) 'measured.dat' via a0,a1,a2,a3
fit a*x + b*y 'surface.dat' using 1:2:3 via a,b
fit [*:*][yaks=*:*] a*x+b*yaks 'surface.dat' u 1:2:3 via a,b


fit [][][t=*:*] a*x + b*y + c*t 'foo.dat' using 1:2:3:4 via a,b,c


set dummy x1, x2, x3, x4, x5
h(x1,x2,x3,x4,s5) = a*x1 + b*x2 + c*x3 + d*x4 + e*x5
fit h(x1,x2,x3,x4,x5) 'foo.dat' using 1:2:3:4:5:6 via a,b,c,d,e
```

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 activated by 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.

If **set fit prescale** is activated, fit parameters are prescaled by their initial values. This helps the Marquardt-Levenberg routine converge more quickly and reliably in cases where parameters differ in size by several orders of magnitude.

The fit may be interrupted by pressing Ctrl-C (Ctrl-Break in wgnuplot). 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 **set fit script** or the environment variable **FIT_SCRIPT**. The default is **replot**, 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 **save fit** command may be used to store final values in a file for subsequent use as a parameter file. See **save fit (p. 128)** for details.

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.

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 (p. 93)** 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/non-linearity 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 a certain limit, see **set fit limit (p. 147)**), or (2) it reaches a preset iteration count limit (see **set fit maxiter (p. 147)**). The fit may also be interrupted and subsequently halted from the keyboard (see **fit (p. 90)**). The user variable FIT_CONVERGED contains 1 if the previous fit command terminated due to convergence; it contains 0 if the previous fit terminated for any other reason. FIT_NITER contains the number of iterations that were done during the last 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 (p. 93)**.

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 **smooth** option to **plot** may be what you've been looking for rather than **fit**.

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'.

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. If the data points have weights, **gnuplot** calculates the so-called p-value, i.e. one minus the cumulative distribution function of the chisquare-distribution for the number of degrees of freedom and the resulting chisquare, see **practical_guidelines (p. 94)**. These values are exported to the variables

```
FIT_NDF = Number of degrees of freedom
FIT_WSSR = Weighted sum-of-squares residual
FIT_STDFIT = sqrt(WSSR/NDF)
FIT_P = p-value
```

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 indicating correlation of parameters in the region of the solution; The main diagonal elements, autocorrelation, are always 1; if all parameters were independent, the off-diagonal 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.

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 favorite 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.

The p-value of the fit is one minus the cumulative distribution function of the chisquare-distribution for the number of degrees of freedom and the resulting chisquare. This can serve as a measure of the goodness-of-fit. The range of the p-value is between zero and one. A very small or large p-value indicates that the model does not describe the data and its errors well. As described above, this might indicate a problem with the data, its errors or the model, or a combination thereof. A small p-value might indicate that the errors have been underestimated and the errors of the final parameters should thus be scaled. See also **set fit errorscaling (p. 147)**.

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.

Control

There are a number of environment variables that can be defined to affect **fit** before starting **gnuplot**, see **fit control environment (p. 96)**. At run time adjustments to the **fit** command operation can be controlled by **set fit**. See **fit control variables (p. 95)**.

Control variables

DEPRECATED in version 5. These user variables used to affect fit behaviour.

```
FIT_LIMIT - use `set fit limit <epsilon>`
FIT_MAXITER - use `set fit maxiter <number_of_cycles>`
```

```
FIT_START_LAMBDA - use `set fit start-lambda <value>`
FIT_LAMBDA_FACTOR - use `set fit lambda-factor <value>`
FIT_SKIP - use the datafile `every` modifier
FIT_INDEX - See `fit multi-branch`
```

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 fit logfile**.

```
FIT_SCRIPT
```

specifies a command that may be executed after an user interrupt. The default is **replot**, but a **plot** or **load** command may be useful to display a plot customized to highlight the progress of the fit. This setting can also be changed using **set fit script**.

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:-2:2:3  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.

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 a false minimum. One way to find starting values is to plot data and the fitting function on the same graph and change parameter values and **replot** until reasonable similarity is reached. The same plot is also useful to check whether the fit found a false minimum.

Of course finding a nice-looking fit does not prove there is no "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.

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 supply the starting parameter values in a 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 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 either 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 or use **set fit prescale** which does this internally according to the parameter starting values.

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!"

Help

The **help** command displays built-in 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.

History

The **history** command prints or saves previous commands in the history list, or reexecutes a previous entry in the list. To modify the behavior of this command, see **set history (p. 155)**.

Input lines with **history** as their first command are not stored in the command history.

Examples:

```
history               # show the complete history
history 5             # show last 5 entries in the history
history quiet 5       # show last 5 entries without entry numbers
history "hist.gp"     # write the complete history to file hist.gp
history "hist.gp" append # append the complete history 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 using pipe
history ?load         # show all history entries starting with "load"
history ?"set c"      # like above, several words enclosed in quotes
hist !"set xr"        # like above, several words enclosed in quotes
hist !55              # reexecute the command at history entry 55
```

If

New syntax:

```
if (<condition>) { <commands>;
       <commands>
       <commands>
} else {
       <commands>
}
```

Old syntax:

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

This version of gnuplot supports block-structured if/else statements. If the keyword **if** or **else** is immediately followed by an opening "{", then conditional execution applies to all statements, possibly on multiple input lines, until a matching "}" terminates the block. If commands may be nested.

The old single-line if/else syntax is still supported, but can not be mixed with the new block-structured syntax. See **if-old (p. 98)**.

If-old

Through gnuplot version 4.4, the scope of the if/else commands was limited to a single input line. Now a multi-line clause may be enclosed in curly brackets. The old syntax is still honored but cannot be used inside a bracketed clause.

If no opening "{" follows the **if** keyword, the command(s) in <command-line> will be executed if <condition> is true (non-zero) or skipped if <condition> is false (zero). Either case will consume commands on the input line until the end of the line or an 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!)

For

The **plot**, **splot**, **set** and **unset** commands may optionally contain an iteration for clause. This has the effect of executing the basic command multiple times, each time re-evaluating any expressions that make use of the iteration control variable. Iteration of arbitrary command sequences can be requested using the **do** command. Two forms of iteration clause are currently supported:
```
for [intvar = start:end{:increment}]
for [stringvar in "A B C D"]
```
Examples:
```
plot for [filename in "A.dat B.dat C.dat"] filename using 1:2 with lines
plot for [basename in "A B C"] basename.".dat" using 1:2 with lines
set for [i = 1:10] style line i lc rgb "blue"
unset for [tag = 100:200] label tag
```
Nested iteration is supported:
```
set for [i=1:9] for [j=1:9] label i*10+j sprintf("%d",i*10+j) at i,j
```
See additional documentation for **iteration (p. 47)**, **do (p. 89)**.

Import

The **import** command associates a user-defined function name with a function exported by an external shared object. This constitutes a plugin mechanism that extends the set of functions available in gnuplot. See **plugins (p. 55)**.

Syntax:
```
import func(x[,y,z,...]) from "sharedobj[:symbol]"
```
Examples:
```
# make the function myfun, exported by "mylib.so" or "mylib.dll"
# available for plotting or numerical calculation in gnuplot
import myfun(x) from "mylib"
import myfun(x) from "mylib:myfun"    # same as above

# make the function theirfun, defined in "theirlib.so" or "theirlib.dll"
# available under a different name
import myfun(x,y,z) from "theirlib:theirfun"
```
The program extends the name given for the shared object by either ".so" or ".dll" depending on the operating system, and searches for it first as a full path name and then as a path relative to the current directory. The operating system itself may also search any directories in LD_LIBRARY_PATH or DYLD_LIBRARY_PATH.

Load

The **load** command executes each line of the specified input file as if it had been typed in interactively. Files created by the **save** 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 **call** commands. See **comments (p. 32)** for information about comments in commands. To **load** with arguments, see **call (p. 86)**.

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 (p. 30)** 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.

Lower

See **raise (p. 126)**.

Pause

The **pause** command displays any text associated with the command and then waits a specified amount of time or until the carriage return is pressed. **pause** is especially useful in conjunction with **load** files.

Syntax:

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

<time> may be any constant or floating-point expression. **pause -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. **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**, **close**, 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. Hotkeys (bind command) are disabled if keypress is one of the end conditions. Zooming is disabled if button3 is one of the end conditions.

In all cases the coordinates of the mouse are returned in variables MOUSE_X, MOUSE_Y, MOUSE_X2, MOUSE_Y2. See **mouse variables (p. 54)**.

Note: Since **pause** 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 active 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 loop such as:

```
print "I will resume after you hit the Tab key in the plot window"
plot <something>
pause mouse key
while (MOUSE_KEY != 9) {
    pause mouse key
}
```

Pause mouse close

The command **pause mouse close** is a specific example of pausing to wait for an external event. In this case the program waits for a "close" event from the plot window. Exactly how to generate such an event varies with your desktop environment and configuration, but usually you can close the plot window by clicking on some widget on the window border or by typing a hot-key sequence such as <alt><F4> or <ctrl>q. If you are unsure whether a suitable widget or hot-key is available to the user, you may also want to define a hot-key sequence using gnuplot's own mechanism. See **bind (p. 52)**.

The command sequence below may be useful when running gnuplot from a script rather than from the command line.

```
plot <...whatever...>
bind all "alt-End" "exit gnuplot"
pause mouse close
```

Plot

plot is the primary command for drawing plots with **gnuplot**. It offers many different graphical representations for functions and data. **plot** is used to draw 2D functions and data. **splot** draws 2D projections of 3D surfaces and data.

Syntax:

```
plot {<ranges>} <plot-element> {, <plot-element>, <plot-element>}
```

Each plot element consists of a definition, a function, or a data source together with optional properties or modifiers:

```
plot-element:
     {<iteration>}
     <definition> | {sampling-range} <function> | <data source>
                  | keyentry
     {axes <axes>} {<title-spec>}
     {with <style>}
```

The graphical representation of each plot element is determined by the keyword **with**, e.g. **with lines** or **with boxplot**. See **plotting styles (p. 61)**.

The data to be plotted is either generated by a function (two functions if in parametric mode), read from a data file, or read from a named data block that was defined previously. Multiple datafiles, data blocks, and/or functions may be plotted in a single plot command separated by commas. See **data (p. 106)**, **inline data (p. 47)**, **functions (p. 119)**.

A plot-element that contains the definition of a function or variable does not create any visible output, see third example below.

Examples:

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

See also **show plot (p. 187)**.

Axes

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).

Binary

BINARY DATA FILES:

It is necessary to provide the keyword **binary** after the filename. Adequate 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, binary matrix format and binary general format.

The **binary matrix** format contains a two dimensional array of 32 bit IEEE float values plus an additional column and row of coordinate values. In the **using** specifier of a plot command, column 1 refers to the matrix row coordinate, column 2 refers to the matrix column coordinate, and column 3 refers to the value stored in the array at those coordinates.

The **binary general** format contains an arbitrary number of columns for which information must be specified at the command line. For example, **array**, **record**, **format** and **using** can indicate the size, format and dimension of data. There are a variety of useful commands for skipping file headers and 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. Unlike reading from a text or matrix binary file, general binary does not treat the generated columns as 1, 2 or 3 in the **using** list. Instead column 1 refers to 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.

Binary matrix is the default binary format when no keywords specific to **binary general** 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 **binary matrix (p. 227)** or **binary general (p. 103)** for more details.

The **index** keyword is not supported, since the file format allows only one surface per file. The **every** and **using** filters are supported. **using** operates as if the data were read in the above triplet form. `Binary File Splot Demo.`

General

The **binary** keyword appearing alone indicates a binary data file that contains both coordinate information describing a non-uniform grid and the value of each grid point (see **binary matrix (p. 227)**). Binary data in any other format requires additional keywords to describe the layout of the data. Unfortunately the syntax of these required additional keywords is convoluted. Nevertheless the general binary mode is particularly useful for application programs sending large amounts of data to gnuplot.

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, non-uniform matrix binary format is assumed. (See **binary matrix (p. 227)** for more details.)

Gnuplot knows how to read a few standard binary file types that are fully self-describing, e.g. PNG images. Type **show datafile binary** at the command line for a list. Apart from these, you can think of binary data files as conceptually the same as text data. Each point has columns of information which are selected via the **using** specification. If no **format** string is specified, gnuplot will read in a number of binary values 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. Certain plot types have an associated default using specification. For example, **with image** has a default of **using 1**, while **with rgbimage** has a default of **using 1:2:3**.

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 of the array. For example, **array=(10,20)** 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 negative number indicates that data should be read until the end of file. If there is only one dimension, the parentheses may be omitted. A colon can be used to separate the dimensions for multiple records. For example, **array=25:35** indicates there are two one-dimensional records in the file.

Record

This keyword serves the same function as **array** and has 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.

Skip

This keyword allows you to skip sections of a binary file. For instance, if the file contains a 1024 byte header before the start of the data region you would probably want to use

```
plot '<file_name>' binary skip=1024 ...
```

If there are multiple records in the file, you may specify a leading offset for each. For example, to skip 512 bytes before the 1st record and 256 bytes before the second and third records

```
plot '<file_name> binary record=356:356:356 skip=512:256:256 ...
```

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, similar to the **using** specification, the format can include discarded columns via the * character and have implicit repetition via a numerical repeat-field. For example, **format="%*2int%3float"** causes gnuplot to discard two ints before reading three floats. 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 (p. 143)**.

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.edfplus.info/specs
```

Png If gnuplot was configured to use the libgd library for png/gif/jpeg output, then it can also be used to read these same image types as binary files. You can use an explicit command

```
plot 'file.png' binary filetype=png
```

Or the file type will be recognized automatically from the extension if you have previously requested

```
set datafile binary filetype=auto
```

Keywords

The following keywords apply only when generating coordinates from binary data files. That is, the control mapping the individual elements of a binary array, matrix, or image to specific x/y/z positions.

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**. I.e. it affects the assignment of pixels to scan lines during input. To instead transpose an image when it is displayed try

```
plot 'imagefile' binary filetype=auto flipx rotate=90deg with rgbimage
```

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 **origin** 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 **origin**, 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 **origin** 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.)

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>}
                   {{nonuniform} matrix}
                   {index <index list> | index "<name>"}
                   {every <every list>}
                   {skip <number-of-lines>}
                   {using <using list>}
                   {smooth <option>}
                   {bins <options>}
                   {volatile} {noautoscale}
```

The modifiers **binary**, **index**, **every**, **skip**, **using**, **bins**, and **smooth** are discussed separately. In brief

- **skip N** tells the program to ignore N lines at the start of the input file
- **binary** indicates that the file contains binary data rather than text
- **index** selects which data sets in a multi-data-set file are to be plotted
- **every** specifies which points within a single data set are to be plotted
- **using** specifies which columns in the file are to be used in which order
- **smooth** performs simple filtering, interpolation, or curve-fitting of the data prior to plotting
- **bins** sorts individual input points into equal-sized intervals along x and plots a single accumulated value per interval

- **volatile** indicates that the content of the file may not be available to reread later and therefore it should be retained internally for re-use.

splot has a similar syntax but does not support **smooth** or **bins**.

The **noautoscale** keyword means that the points making up this plot will be ignored when automatically determining axis range limits.

TEXT DATA FILES:

Each non-empty line in a data file describes one data point, except that records beginning with # (and also with **!** on VMS) will be treated as comments and ignored.

Depending on the plot style and options selected, from one to eight values are read from each line and associated with a single data point. See **using (p. 115)**.

The individual records on a single line of data must be separated by white space (one or more blanks or tabs) a special field separator character is is specified by the **set datafile** command. A single field may itself contain white space characters if the entire field is enclosed in a pair of double quotes, or if a field separator other than white space is in effect. 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 or E. The fortran exponential specifiers d, D, q, and Q may also be used if the command **set datafile fortran** is in effect.

Blank records in a data file are significant. 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). Two blank records in a row indicate a break between separate data sets. See **index (p. 110)**.

If autoscaling has been enabled (**set autoscale**), 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 **noextend** modifier of the **set autoscale** or **set [axis]range** commands. This turns off extension of the axis range to include the next tic mark.

Label coordinates and text can also be read from a data file (see **labels (p. 75)**).

Bins

Syntax:

```
plot 'DATA' using <XCOL> {:<YCOL>} bins{=<NBINS>}
     {binrange [<LOW>:<HIGH>]} {binwidth=<width>}
     {binvalue={sum|avg}
```

The **bins** option to a **plot** command first assigns the original data to equal width bins on x and then plots a single value per bin. The default number of bins is controlled by **set samples**, but this can be changed by giving an explicit number of bins in the command.

If no binrange is given, the range is taken from the extremes of the x values found in 'DATA'.

Given the range and the number of bins, bin width is calculated automatically and points are assigned to bins 0 to NBINS-1

```
BINWIDTH = (HIGH - LOW) / (NBINS-1)
xmin = LOW - BINWIDTH/2
xmax = HIGH + BINWIDTH/2
first bin holds points with (xmin <= x < xmin + BINWIDTH)
last bin holds points with (xmax-BINWIDTH <= x < xman)
each point is assigned to bin i = floor(NBINS * (x-xmin)/(xmax-xmin))
```

Alternatively you can provide a fixed bin width, in which case nbins is calculated as the smallest number of bins that will span the range.

On output bins are plotted or tabulated by midpoint. E.g. if the program calculates bin width as shown above, the x coordinate output for the first bin is x=LOW (not x=xmin).

If only a single column is given in the using clause then each data point contributes a count of 1 to the accumulation of total counts in the bin for that x coordinate value. If a second column is given then the value in that column is added to the accumulation for the bin. Thus the following two plot command are equivalent:

```
plot 'DATA" using N bins=20
set samples 20
plot 'DATA' using (column(N)):(1)
```

The y value plotted for each bin is the sum of the y values over all points in that bin. This corresponds to **binvalue=sum**. EXPERIMENTAL: **binvalue=avg** instead plots the mean y value for that bin.

For related plotting styles see **smooth frequency (p. 112)** and **smooth kdensity (p. 113)**.

Columnheaders

Extra lines at the start of a data file may be explicitly ignored using the **skip** keyword in the plot command. A single additional line containing text column headers may be present. It is skipped automatically if the plot command refers explicitly to column headers, e.g. by using them for titles. Otherwise you may need to skip it explicitly either by adding one to the skip count or by setting the attribute **set datafile columnheaders**. See **skip (p. 111)**, **columnhead (p. 39)**, **autotitle columnheader (p. 159)**, **set datafile (p. 141)**.

Csv files

Syntax:

```
set datafile separator {whitespace | tab | comma | "chars"}
```

"csv" is short for "comma-separated values". The term "csv file" is loosely applied to files in which data fields are delimited by a specific character, not necessarily a comma. To read data from a csv file you must tell gnuplot what the field-delimiting character is. For instance to read from a file using semicolon as a field delimiter:

```
set datafile separator ";"
```

See **set datafile separator (p. 142)**. This applies only to files used for input. To create a csv file on output, use the corresponding **separator** option to **set table**.

Every

The **every** keyword allows a periodic sampling of a data set to be plotted.

For ordinary files a "point" single record (line); a "block" of data is a set of consecutive records with blank lines before and after the block.

For matrix data a "block" and "point" correspond to "row" and "column". See **matrix every (p. 228)**.

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. ':' at the end of the **every** option is not permitted. If **every** 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 every 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

.

Example datafile

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

```
pop(x) = 103*exp((1965-x)/10)
set xrange [1960:1990]
plot '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
```

Binary examples:

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

```
# The data file header contains all details necessary 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 components of a raw RGB image
# and flips the y-dimension so that typical image orientation (start
# at top left corner) translates to the Cartesian plane.  Pixel
# spacing is given and there are two images in the file.  One of them
# is translated via origin.
plot '<file_name>' binary array=(512,1024):(1024,512) format='%uchar' \
     dx=2:1 dy=1:2 origin=(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 different from
# the system on which gnuplot is running.
splot '<file_name>' binary record=30:30:29:26 endian=swap u 1:2:3

# Same input file, but this time we skip the 1st and 3rd records
splot '<file_name>' binary record=30:26 skip=360:348 endian=swap u 1:2:3
```

See also **binary matrix (p. 227)**.

Index

The **index** keyword allows you to select specific data sets in a multi-data-set file for plotting.

Syntax:

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

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 <p> is specified but <n> is left blank then every <p>-th dataset is read until the end of the file. If **index** is not specified, the entire file is plotted as a single data set.

Example:

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

For each point in the file, the index value of the data set it appears in is available via the pseudo-column **column(-2)**. This leads to an alternative way of distinguishing individual data sets within a file as shown below. This is more awkward than the **index** command if all you are doing is selecting one data set for plotting, but is very useful if you want to assign different properties to each data set. See **pseudocolumns (p. 116)**, **lc variable (p. 50)**.

Example:

```
plot 'file' using 1:(column(-2)==4 ? $2 : NaN)        # very awkward
plot 'file' using 1:2:(column(-2)) linecolor variable # very useful!
```

index '<name>' selects the data set with name '<name>'. Names are assigned to data sets in comment lines. The comment character and leading white space are removed from the comment line. If the resulting line starts with <name>, the following data set is now named <name> and can be selected.

Example:

```
plot 'file' index 'Population'
```

Please note that every comment that starts with <name> will name the following data set. To avoid problems it may be useful to choose a naming scheme like '== Population ==' or '[Population]'.

Skip

The **skip** keyword tells the program to skip lines at the start of a text (i.e. not binary) data file. The lines that are skipped do not count toward the line count used in processing the **every** keyword. Note that **skip N** skips lines only at the start of the file, whereas **every ::N** skips lines at the start of every block of data in the file. See also **binary skip (p. 103)** for a similar option that applies to binary data files.

Smooth

gnuplot includes a few general-purpose routines for filtering, interpolation and grouping data as it is input; these are grouped under the **smooth** 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 | fnormal | cumulative | cnormal | bins
               | kdensity {bandwidth} {period}
               | csplines | acsplines | mcsplines | bezier | sbezier
               | unwrap | zsort}
```

The **unique**, **frequency**, **fnormal**, **cumulative** and **cnormal** sort the data on x and then plot some aspect of the distribution of x values.

The spline and Bezier options determine coefficients describing 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 **set samples (p. 194)**) and connecting these points with straight line segments. If the data set is interrupted by blank lines or undefined values a separate continuous curve is fit for each uninterrupted subset of the data. Adjacent separately fit segments may be separated by a gap or discontinuity.

unwrap manipulates the data to avoid jumps of more than pi by adding or subtracting multiples of 2*pi.

zsort uses a 3rd column of input to sort points prior to plotting.

If **autoscale** is in effect, axis ranges will be computed for the final curve rather than for the original data.

If **autoscale** is not in effect, and a spline curve is being generated, sampling of the spline fit is done across the intersection of the x range covered by the input data and the fixed abscissa range defined by **set xrange**.

If too few points are available to apply the requested smoothing operation an error message is produced.

The **smooth** options have no effect on function plots.

Acsplines The **smooth acsplines** option approximates the data with a natural smoothing spline. After the data are made monotonic in x (see **smooth unique (p. 112)**), a curve is piecewise constructed from segments of cubic polynomials whose coefficients are found by fitting to the individual data points weighted by the value, if any, given in the third column of the using spec. The default is equivalent to

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

Bins **smooth bins** is the same as **bins**. See **bins (p. 107)**. For related plotting styles see **smooth frequency (p. 112)** and **smooth kdensity (p. 113)**.

Csplines The **smooth csplines** option connects consecutive points by natural cubic splines after rendering the data monotonic (see **smooth unique (p. 112)**).

Mcsplines The **smooth mcsplines** option connects consecutive points by cubic splines constrained such that the smoothed function preserves the monotonicity and convexity of the original data points. This reduces the effect of outliers. FN Fritsch & RE Carlson (1980) "Monotone Piecewise Cubic Interpolation", SIAM Journal on Numerical Analysis 17: 238–246.

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

Unique The **smooth 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.

Unwrap The **smooth unwrap** option modifies the input data so that any two successive points will not differ by more than pi; a point whose y value is outside this range will be incremented or decremented by multiples of 2pi until it falls within pi of the previous point. This operation is useful for making wrapped phase measurements continuous over time.

Frequency The **smooth 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. To plot a histogram of the number of data values in equal size bins, set the y-value to 1.0 so that the sum is a count of occurrences in that bin. This is done implicitly if only a single column is provided. Example:

```
binwidth = <something>  # set width of x values in each bin
bin(val) = binwidth * floor(val/binwidth)
plot "datafile" using (bin(column(1))):(1.0) smooth frequency
plot "datafile" using (bin(column(1))) smooth frequency  # same result
```

See also `smooth.dem`

Fnormal The **smooth fnormal** option work just like the **frequency** option, but produces a normalized histogram. It makes the data monotonic in x and normalises the y-values so they all sum to 1. Points with the same x-value are replaced by a single point containing the sumed y-values. To plot a histogram of the number of data values in equal size bins, set the y-value to 1.0 so that the sum is a count of occurrences in that bin. This is done implicitly if only a single column is provided. See also `smooth.dem`

Cumulative The **smooth cumulative** option makes the data monotonic in x; points with the same x-value are replaced by a single point containing the cumulative sum of y-values of all data points with lower x-values (i.e. to the left of the current data point). This can be used to obtain a cumulative distribution function from data. See also `smooth.dem`

Cnormal The **smooth cnormal** option makes the data monotonic in x and normalises the y-values onto the range [0:1]. Points with the same x-value are replaced by a single point containing the cumulative sum of y-values of all data points with lower x-values (i.e. to the left of the current data point) divided by the total sum of all y-values. This can be used to obtain a normalised cumulative distribution function from data (useful when comparing sets of samples with differing numbers of members). See also `smooth.dem`

Kdensity The **smooth kdensity** option generates and plots a kernel density estimate using Gaussian kernels for the distribution from which a set of values was drawn. Values are taken from the first data column, optional weights are taken from the second column. A Gaussian is placed at the location of each point and the sum of all these Gaussians is plotted as a function. To obtain a normalized histogram, each weight should be 1/number-of-points.

Bandwidth: By default gnuplot calculates and uses the bandwidth which would be optimal for normally distributed data values.

```
default_bandwidth = sigma * (4/3N) ** (0.2)
```

This will usually be a very conservative, i.e. broad bandwidth. Alternatively, you can provide an explicit bandwidth.

```
plot $DATA smooth kdensity bandwidth <value> with boxes
```

The bandwidth used in the previous plot is stored in GPVAL_KDENSITY_BANDWIDTH.

Period: For periodic data individual Gaussian components should be treated as repeating at intervals of one period. One example is data measured as a function of angle, where the period is 2pi. Another example is data indexed by day-of-year and measured over multiple years, where the period is 365. In such cases the period should be provided in the plot command:

```
plot $ANGULAR_DAT smooth kdensity period 2*pi with lines
```

Zsort Syntax

```
plot FOO using x:y:z:color smooth zsort with points lc palette
```

The intended use is to filter presentation of 2D scatter plots with a huge number of points so that the distribution of high-scoring points remains evident. Sorting the points on z guarantees that points with a high z-value will not be obscured by points with lower z-values. Limited to plot style "with points".

Special-filenames

There are a few filenames that have a special meaning: '', '-', '+' and '++'.

The empty filename '' tells gnuplot to re-use the previous input file in the same plot command. So to plot two columns from the same input file:

```
plot 'filename' using 1:2, '' using 1:3
```

The filename can also be reused over subsequent plot commands, however **save** then only records the name in a comment.

The special filenames '+' and '++' are a mechanism to allow the full range of **using** specifiers and plot styles with inline functions. Normally a function plot can only have a single y (or z) value associated with each sampled point. The pseudo-file '+' treats the sampled points as column 1, and allows additional column values to be specified via a **using** specification, just as for a true input file. The number of samples is controlled via **set samples**. By default samples are generated over the range given by **set trange**, or if trange has not been set than over the full range of **set xrange**.

Note: The use of trange is a change from previous gnuplot versions. It allows the sampling range to differ from the x axis range.

```
plot '+' using ($1):(sin($1)):(sin($1)**2) with filledcurves
```

An independent sampling range can be provided immediately before the '+'. As in normal function plots, a name can be assigned to the independent variable. If given for the first plot element, the sampling range specifier has to be preceded by the **sample** keyword (see also **plot sampling (p. 121)**).

```
plot sample [beta=0:2*pi] '+' using (sin(beta)):(cos(beta)) with lines
```

Additionally, the range specifier of '+' supports giving a sampling increment.

```
plot $MYDATA, [t=-3:25:1] '+' using (t):(f(t))
```

The pseudo-file '++' returns 2 columns of data forming a regular grid of [u,v] coordinates with the number of points along u controlled by **set samples** and the number of points along v controlled by **set isosamples**. You must set urange and vrange before plotting '++'. However the x and y ranges can be autoscaled or can be explicitly set to different values than urange and vrange. Use of u and v to sample '++' is a CHANGE introduced in version 5.2 Examples:

```
splot '++' using 1:2:(sin($1)*sin($2)) with pm3d
plot '++' using 1:2:(sin($1)*sin($2)) with image
```

The special filename **'-'** 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.

'-' is intended for situations where it is useful to have data and commands together, e.g. when both are piped to **gnuplot** from another application. Some of the demos, for example, might use this feature. While **plot** options such as **index** and **every** are recognized, their use forces you to enter data that won't be used. For all but the simplest cases it is probably easier to first define a datablock and then read from it rather than from **'-'**. See **datablocks (p. 47)**.

If you use **'-'** with **replot**, you may need to enter the data more than once. See **replot (p. 127)**, **refresh (p. 127)**. Here again it may be better to use a datablock.

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, '' using 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.)

Piped-data

On systems with a popen function, 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 **using** keyword.

On systems with an fdopen() function, data can be read from an arbitrary file descriptor attached to either a file or pipe. To read from file descriptor **n** use **'<&n'**. This allows you to easily pipe in several data files in a single call from a POSIX shell:

```
$ gnuplot -p -e "plot '<&3', '<&4'" 3<data-3 4<data-4
$ ./gnuplot 5< <(myprogram -with -options)
gnuplot> plot '<&5'
```

Using

The most common datafile modifier is **using**. It tells the program which columns of data in the input file are to be plotted.

Syntax:

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

If a format is specified, it is used to read in each datafile record using the C library 'scanf' function. Otherwise the record is interpreted as consisting of columns (fields) of data separated by whitespace (spaces and/or tabs), but see **datafile separator (p. 142)**.

Each <entry> may be a simple column number that selects the value from one field of the input file, a string that matches a column label in the first line of a data set, an expression enclosed in parentheses, or a special function not enclosed in parentheses such as xticlabels(2).

If the entry is an expression in parentheses, then the function column(N) may be used to indicate the value in column N. That is, column(1) refers to the first item read, column(2) to the second, and so on. The special symbols $1, $2, ... are shorthand for column(1), column(2) ...

The special symbol $# evaluates to the total number of columns in the current line of input, so column($#) or stringcolumn($#) always returns the content of the final column even if the number of columns is unknown or different lines in the file contain different numbers of columns.

The function **valid(N)** tests whether column N contains a valid number. If each column of data in the input file contains a label in the first row rather than a data value, this label can be used to identify the column on input and/or in the plot legend. The column() function can be used to select an input column by label rather than by column number. For example, if the data file contains

```
Height    Weight    Age
val1      val1      val1
...       ...       ...
```

then the following plot commands are all equivalent

```
plot 'datafile' using 3:1, '' using 3:2
plot 'datafile' using (column("Age")):(column(1)), \
              '' using (column("Age")):(column(2))
plot 'datafile' using "Age":"Height", '' using "Age":"Weight"
```

The full string must match. Comparison is case-sensitive. To use column labels in the plot legend, use **set key autotitle columnhead**.

In addition to the actual columns 1...N in the input data file, gnuplot presents data from several "pseudo-columns" that hold bookkeeping information. E.g. $0 or column(0) returns the sequence number of this data record within a dataset. Please see **pseudocolumns (p. 116)**.

An empty <entry> will default to its order in the list of entries. For example, **using ::4** is interpreted as **using 1:2:4**.

If the **using** list has only a single entry, that <entry> will be used for y and the data point number (pseudo-column $0) is used for x; for example, "**plot 'file' using 1**" is identical to "**plot 'file' using 0:1**". If the **using** list has two entries, these will be used for x and y. See **set style (p. 196)** and **fit (p. 90)** for details about plotting styles that make use of data from additional columns of input.

'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.

Using_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 comma**.

```
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. Or you can use the pre-defined variable NaN to achieve the same result.

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 (a) **plot 'file'**, (b) **plot 'file' using 1:2**, and (c) **plot 'file' using ($1):($2)** can be subtly different. The exact behaviour has changed in version 5. See **missing (p. 141)**.

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.

Pseudocolumns Expressions in the **using** clause of a plot statement can refer to additional bookkeeping values in addition to the actual data values contained in the input file. These are contained in "pseudo-columns".

```
column(0)   The sequential order of each point within a data set.
            The counter starts at 0, increments on each non-blank,
            non-comment line, and is reset by two sequential blank
            records.  The shorthand form $0 is available.
column(-1)  This counter starts at 0, increments on a single blank line,
            and is reset by two sequential blank lines.
```

```
                This corresponds to the data line in array or grid data.
                It can also be used to distinguish separate line segments
                or polygons within a data set.
column(-2)      Starts at 0 and increments on two sequential blank lines.
                This is the index number of the current data set within a
                file that contains multiple data sets.  See 'index'.
column($#)      The special symbol $# evaluates to the total number of
                columns available, so column($#) refers to the last
                (rightmost) field in the current input line.
                column($# - 1) would refer to the last-but-one column, etc.
```

Key The layout of certain plot styles (column-stacked histograms, spider plots) is such that it would make no sense to generate plot titles from a data column header. Also it would make no sense to generate axis tic labels from the content of a data column (e.g. **using 2:3:xticlabels(1)**). These plots styles instead use the form **using 2:3:key(1)** to generate plot titles for the key from the text content of a data column, usually a first column of row headers. See the example given for **spiderplot (p. 79)**.

Xticlabels Axis tick labels can be generated via a string function, usually taking a data column as an argument. The simplest form uses the data column itself as a string. That is, xticlabels(N) is shorthand for xticlabels(stringcolumn(N)). This example uses the contents of column 3 as x-axis tick labels.

```
plot 'datafile' using <xcol>:<ycol>:xticlabels(3) with <plotstyle>
```

Axis tick labels may be generated 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 **using** portion of the command. For each data point which has a valid set of X,Y[,Z] coordinates, the string value given to xticlabels() is added to the list of xtic labels at the same X coordinate as the point it belongs to. **xticlabels()** may be shortened to **xtic()** and so on.

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.

Example:

```
plot "data" using 1:2:xtic( $3 > 10. ? "A" : "B" )
```

This example shows the use of a string-valued function to generate x-axis tick labels. Each point in the data file generates a tick mark on x labeled either "A" or "B" depending on the value in column 3.

X2ticlabels See **plot using xticlabels (p. 117)**.

Yticlabels See **plot using xticlabels (p. 117)**.

Y2ticlabels See **plot using xticlabels (p. 117)**.

Zticlabels See **plot using xticlabels (p. 117)**.

Cbticlabels EXPERIMENTAL (details may change in a future release version) 2D plots: colorbar labels are placed at the palette coordinate used by the plot for variable coloring "lc palette z". 3D plots: colorbar labels are placed at the z coordinate of the point. Note that in the case of a 3D heat map with variable color that does not match z, this is probably not the correct label. See also **plot using xticlabels (p. 117)**.

Volatile

The **volatile** keyword in a plot command indicates that the data previously read from the input stream or file may not be available for re-reading. This tells the program to use **refresh** rather than **replot** commands whenever possible. See **refresh (p. 127)**.

Errorbars

Error bars are supported for 2D data file plots by reading one to four additional columns (or **using** 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 **using** 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 **using** 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 **errorlines (p. 118)**).

The tic marks at the ends of the bar are controlled by **set errorbars**.

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

See also `errorbar demos.`

See **plot using (p. 115)**, **plot with (p. 124)**, and **set style (p. 196)** for more information.

Errorlines

Lines with error bars are supported for 2D data file plots by reading one to four additional columns (or **using** 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 **using** 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 **using** 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 tic marks at the ends of the bar are controlled by **set errorbars**.

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

See **plot using (p. 115)**, **plot with (p. 124)**, and **set style (p. 196)** for more information.

Functions

Built-in or user-defined functions can be displayed by the **plot** and **splot** commands in addition to, or instead of, data read from a file. The requested function is evaluated by sampling at regular intervals spanning the independent axis range[s]. See **set samples (p. 194)** and **set isosamples (p. 156)**. Example:

```
approx(ang) = ang - ang**3 / (3*2)
plot sin(x) title "sin(x)", approx(x) title "approximation"
```

To set a default plot style for functions, see **set style function (p. 200)**. For information on built-in functions, see **expressions functions (p. 37)**. For information on defining your own functions, see **user-defined (p. 43)**.

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 **with** 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.`

Ranges

This section describes only the optional axis ranges that may appear as the very first items in a **plot** command. If present, these ranges override any range limits established by a previous **set range** statement. For optional ranges elsewhere in a **plot** command that limit sampling of an individual plot component see **sampling (p. 121)**.

Syntax:

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

The first form applies to the independent variable (**xrange** or **trange**, if in parametric mode). The second form applies to dependent variables. <dummy-var> optionally establishes a new name for the independent variable. (The default name may be changed with **set dummy**.)

In non-parametric mode, ranges must be given in the order

```
plot [<xrange>][<yrange>][<x2range>][<y2range>] ...
```

In parametric mode, ranges must be given in the order

```
plot [<trange>][<xrange>][<yrange>][<x2range>][<y2range>] ...
```

The following **plot** command shows setting **trange** to [-pi:pi], **xrange** to [-1.3:1.3] and **yrange** to [-1:1] for the duration of the graph:

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

* can be used to allow autoscaling of either of min and max. Use an empty range [] as a placeholder if necessary.

Ranges specified on the **plot** or **splot** command line affect only that one graph; use the **set xrange**, **set yrange**, etc., commands to change the default ranges for future graphs.

The use of on-the-fly range specifiers in a plot command may not yield the expected result for linked axes (see **set link (p. 165)**).

For time data you must provide the range in quotes, using the same format used to read time from the datafile. See **set timefmt (p. 208)**.

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:

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

This sets xmax and ymin only:

```
plot [:200] [-pi:]  $mydata using 1:2
```

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'
```

Sampling

1D sampling (x or t axis)

By default, computed functions or data generated for the pseudo-file "+" are sampled over the entire range of the plot as set by a prior **set xrange** command, by an explicit global range specifier at the very start of the plot or splot command, or by autoscaling the xrange to span data seen in all the elements of this plot. However, individual plot components can be assigned a more restricted sampling range.

Examples:

This establishes a total range on x running from 0 to 1000 and then plots data from a file and two functions each spanning a portion of the total range:

```
plot [0:1000] 'datafile', [0:200] func1(x), [200:500] func2(x)
```

This is similar except that the total range is established by the contents of the data file. In this case the sampled functions may or may not be entirely contained in the plot:

```
set autoscale x
plot 'datafile', [0:200] func1(x), [200:500] func2(x)
```

This command is ambiguous. The initial range will be interpreted as applying to the entire plot, not solely to the sampling of the first function as was probably the intent:

```
plot [0:10] f(x), [10:20] g(x), [20:30] h(x)
```

This command removes the ambiguity of the previous example by inserting the keyword **sample** so that the range is not applied to the entire plot:

```
plot sample [0:10] f(x), [10:20] g(x), [20:30] h(x)
```

This example shows one way of tracing out a helix in a 3D plot

```
splot [-2:2][-2:2] sample [h=1:10] '+' using (cos(h)):(sin(h)):(h)
```

2D sampling (u and v axes)

Computed functions or data generated for the pseudo-file '++' use samples generated along the u and v axes. This is a CHANGE from versions prior to 5.2 which sampled along the x and y axes. See **special-filenames ++ (p. 113)**. 2D sampling can be used in either **plot** or **splot** commands.

Example of 2D sampling in a 2D **plot** command. These commands generated the plot shown for plotstyle **with vectors**. See **vectors (p. 80)**.

```
set urange [ -2.0 : 2.0 ]
set vrange [ -2.0 : 2.0 ]
plot '++' using ($1):($2):($2*0.4):(-$1*0.4) with vectors
```

Example of 2D sampling in a 3D **splot** command. These commands are similar to the ones used in **sampling.dem**. Note that the two surfaces are sampled over u and v ranges smaller than the full x and y ranges of the resulting plot.

```
set title "3D sampling range distinct from plot x/y range"
set xrange [1:100]
set yrange [1:100]
splot sample [u=30:70][v=0:50] '++' using 1:2:(u*v) lt 3, \
      [u=40:80][v=30:60] '++' using (u):(v):(u*sqrt(v)) lt 4
```

The range specifiers for sampling on u and v can include an explicit sampling interval to control the number and spacing of samples:

```
splot sample [u=30:70:1][v=0:50:5] '++' using 1:2:(func($1,$2))
```

For loops in plot command

If many similar files or functions are to be plotted together, it may be convenient to do so by iterating over a shared plot command.

Syntax:

```
plot for [<variable> = <start> : <end> {:<increment>}]
plot for [<variable> in "string of words"]
```

The scope of an iteration ends at the next comma or the end of the command, whichever comes first. An exception to this is that definitions are grouped with the following plot item even if there is an intervening comma. Note that iteration does not work for plots in parametric mode.

Example:

```
plot for [j=1:3] sin(j*x)
```

Example:

```
plot for [dataset in "apples bananas"] dataset."dat" title dataset
```

In this example iteration is used both to generate a file name and a corresponding title.

Example:

```
file(n) = sprintf("dataset_%d.dat",n)
splot for [i=1:10] file(i) title sprintf("dataset %d",i)
```

This example defines a string-valued function that generates file names, and plots ten such files together. The iteration variable ('i' in this example) is treated as an integer, and may be used more than once.

Example:

```
set key left
plot for [n=1:4] x**n sprintf("%d",n)
```

This example plots a family of functions.

Example:

```
list = "apple banana cabbage daikon eggplant"
item(n) = word(list,n)
plot for [i=1:words(list)] item[i].".dat" title item(i)
list = "new stuff"
replot
```

This example steps through a list and plots once per item. Because the items are retrieved dynamically, you can change the list and then replot.

Example:

```
list = "apple banana cabbage daikon eggplant"
plot for [i in list] i.".dat" title i
list = "new stuff"
replot
```

This example does exactly the same thing as the previous example, but uses the string iterator form of the command rather than an integer iterator.

If an iteration is to continue until all available data is consumed, use the symbol * instead of an integer <end>. This can be used to process all columns in a line, all datasets (separated by 2 blank lines) in a file, or all files matching a template.

Examples:

```
plot for [i=2:*] 'datafile' using 1:i with histogram
splot for [i=0:*] 'datafile' index i using 1:2:3 with lines
plot for [i=1:*] file=sprintf("File_%03d.dat",i) file using 2 title file
```

Title

By default each plot is listed in the key by the corresponding function or file name. You can give an explicit plot title instead using the **title** option.

Syntax:

```
title <text> | notitle [<ignored text>]
title columnheader | title columnheader(N)
      {at {beginning|end}} {{no}enhanced}
```

where <text> is a quoted string or an expression that evaluates to a string. The quotes will not be shown in the key. Note: Starting with gnuplot version 5.4, if <text> is an expression or function it it evaluated after the corresponding function or data stream is plotted. This allows the title to reference quantities calculated or input during plotting, which was not possible in earlier gnuplot versions.

There is also an option that will interpret the first entry in a column of input data (i.e. the column header) as a text field, and use it as the key title. See **datastrings (p. 33)**. This can be made the default by specifying **set key autotitle columnhead**.

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 using **set key (p. 157)**.

The **at** keyword allows you to place the plot title somewhere outside the auto-generated key box. The title can be placed immediately before or after the line in the graph itself by using **at {beginning|end}**. This option may be useful when plotting **with lines** but makes little sense for most other styles.

To place the plot title at an arbitrary location on the page, use the form **at <x-position>,<y-position>**. By default the position is interpreted in screen coordinates; e.g. **at 0.5, 0.5** is always the middle of the screen regardless of plot axis scales or borders. The format of titles placed in this way is still affected by key options. See **set key (p. 157)**.

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"
```

Plot multiple columns of data, each of which contains its own title on the first line of the file. Place the titles after the corresponding lines rather than in a separate key:

```
unset key
set offset 0, graph 0.1
plot for [i=1:4] 'data' using i with lines title columnhead at end
```

Create a single key area for two separate plots:

```
set key Left reverse
set multiplot layout 2,2
plot sin(x) with points pt 6 title "Left plot is sin(x)" at 0.5, 0.30
plot cos(x) with points pt 7 title "Right plot is cos(x)" at 0.5, 0.27
unset multiplot
```

With

Functions and data may be displayed in one of a large number of styles. The **with** 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>}
                  {arrowstyle | as <arrowstyle_index>}
                  {fill | fs <fillstyle>} {fillcolor | fc <colorspec>}
                  {nohidden3d} {nocontours} {nosurface}
                  {palette}}
             }
```

where <style> is one of

```
lines       dots     steps    vectors     yerrorlines
points      impulses fsteps   xerrorbar   xyerrorbars
linespoints labels   histeps  xerrorlines xyerrorlines
financebars surface  arrows   yerrorbar   parallelaxes
```

or

```
boxes        boxplot      ellipses     histograms rgbalpha
boxerrorbars candlesticks filledcurves image      rgbimage
boxxyerror   circles      fillsteps    pm3d       polygons
isosurface   zerrorfill
```

or

```
table
```

The first group of styles have associated line, point, and text properties. The second group of styles also have fill properties. See **fillstyle (p. 199)**. Some styles have further sub-styles. See **plotting styles (p. 61)** for details of each. The **table** style produces tabular output rather than a plot. See **set table (p. 204)**.

A default style may be chosen by **set style function** and **set style data**.

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. To see the complete set of line and point types available for the current terminal, type **test (p. 233)**.

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 **test** 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 **set pointsize (p. 192)** for details. But note that both <point_size> as set here and as set by **set pointsize** 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.

Both 2D and 3D plots (**plot** and **splot** commands) can use colors from a smooth palette set previously with the command **set palette**. The color value corresponds to the z-value of the point itself or to a separate color coordinate provided in an optional additional **using** colymn. Color values may be treated either as a fraction of the palette range (**palette frac**) or as a coordinate value mapped onto the colorbox range (**palette** or **palette z**). See **colorspec (p. 49)**, **set palette (p. 180)**, **linetype (p. 164)**.

The keyword **nohidden3d** applies only to plots made with the **splot** command. Normally the global option **set hidden3d** 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 other than surfaces (i.e. lines, dots, labels, ...) of a plot marked **nohidden3d** will all be drawn, even if they would normally be obscured by other plot elements.

Similarly, the keyword **nocontours** will turn off contouring for an individual plot even if the global property **set contour** is active.

Similarly, the keyword **nosurface** will turn off the 3D surface for an individual plot even if the global property **set surface** is active.

The keywords may be abbreviated as indicated.

Note that the **linewidth**, **pointsize** and **palette** 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 y=10
```

This plots x*x and a color box:

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

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 (p. 36)**. The output file can be set with **set print**. See also **printerr (p. 126)**.

Printerr

printerr is the same as print except that output is always sent to stderr even if a prior **set print** command remains in effect.

Pwd

The **pwd** command prints the name of the working directory to the screen.

Note that if you wish to store the current directory into a string variable or use it in string expressions, then you can use variable GPVAL_PWD, see **show variables all (p. 210)**.

Quit

The **exit** and **quit** commands and END-OF-FILE character will exit **gnuplot**. Each of these commands will clear the output device (as does the **clear** command) before exiting.

Raise

Syntax:

```
raise {plot_window_id}
lower {plot_window_id}
```

The **raise** and **lower** commands function only for a some terminal types and may depend also on your window manager and display preference settings. An example of use is shown here

```
set term wxt 123     # create first plot window
plot $FOO
lower                # lower the only plot window that exists so far
set term wxt 456     # create 2nd plot window may occlude the first one
plot $BAZ
raise 123            # raise first plot window
```

These commands are known to be unreliable.

Refresh

The **refresh** command is similar to **replot**, with two major differences. **refresh** reformats and redraws the current plot using the data already read in. This means that you can use **refresh** for plots with inline data (pseudo-device '-') and for plots from datafiles whose contents are volatile. You cannot use the **refresh** command to add new data to an existing plot.

Mousing operations, in particular zoom and unzoom, will use **refresh** rather than **replot** if appropriate. Example:

```
plot 'datafile' volatile with lines, '-' with labels
100 200 "Special point"
e
# Various mousing operations go here
set title "Zoomed in view"
set term post
set output 'zoom.ps'
refresh
```

Replot

The **replot** 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 **replot** command will be added onto the last **plot** or **splot** command (with an implied ',' separator) before it is repeated. **replot** accepts the same arguments as the **plot** and **splot** commands except that ranges cannot be specified. Thus you can use **replot** 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, because it will require that you type in the data all over again. In most cases you can use the **refresh** command instead, which will redraw the plot using the data previously read in.

Note that in multiplot mode, **replot** can only reproduce the most recent component plot, not the full set.

See also **command-line-editing (p. 32)** for ways to edit the last **plot (p. 101)** (**splot (p. 225)**) command.

See also **show plot (p. 187)** to show the whole current plotting command, and the possibility to copy it into the **history (p. 98)**.

Reread

[DEPRECATED in version 5.4]

This command is deprecated in favor of explicit iteration. See **iterate (p. 47)**. The **reread** command causes the current **gnuplot** command file, as specified by a **load** command, 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 **reread** command. The **reread** command has no effect when reading interactively (from stdin).

Reset

```
reset {bind | errors | session}
```

The **reset** command causes all graph-related options that can be set with the **set** command to return to their default values. This command can be used to restore the default settings after executing a loaded command file, or to return to a defined state after lots of settings have been changed.

The following are *not* affected by **reset**:

```
‘set term‘ ‘set output‘ ‘set loadpath‘ ‘set linetype‘ ‘set fit‘
‘set encoding‘ ‘set decimalsign‘ ‘set locale‘ ‘set psdir‘
‘set overflow‘ ‘set multiplot‘
```

Note that **reset** does not necessarily return settings to the state they were in at program entry, because the default values may have been altered by commands in the initialization files gnuplotrc or $HOME/.gnuplot. However, these commands can be re-executed by using the variant command **reset session**.

reset session deletes any user-defined variables and functions, restores default settings, and then re-executes the system-wide gnuplotrc initialization file and any private $HOME/.gnuplot preferences file. See **initialization (p. 55)**.

reset errors clears only the error state variables GPVAL_ERRNO and GPVAL_ERRMSG.

reset bind restores all hotkey bindings to their default state.

Save

Syntax:

```
save  {functions | variables | terminal | set | fit} '<filename>'
```

If no option is specified, **gnuplot** saves functions, variables, **set** options and the last **plot** (**splot**) command.

saved files are written in text format and may be read by the **load** command. For **save** with the **set** option or without any option, the **terminal** choice and the **output** 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 (p. 205)**.

save fit saves only the variables used in the most recent **fit** command. The saved file may be used as a parameter file to initialize future fit commands using the **via** keyword.

The filename must be enclosed in quotes.

The special filename "-" may be used to **save** 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 (p. 30)** 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'
```

Set-show

The **set** command can be used to set *lots* of options. No screen is drawn, however, until a **plot**, **splot**, or **replot** 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 **unset** command. See also the **reset (p. 128)** command, which returns all settable parameters to default values.

The **set** and **unset** commands may optionally contain an iteration clause. See **plot for (p. 122)**.

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 3D mapping as set via the **set mapping** 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 **set angles**.

set angles 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 set angles demo.`

Arrow

Arbitrary arrows can be placed on a plot using the **set arrow** command.

Syntax:

```
set arrow {<tag>} from <position> to <position>
set arrow {<tag>} from <position> rto <position>
set arrow {<tag>} from <position> length <coord> angle <ang>
set arrow <tag> arrowstyle | as <arrow_style>
set arrow <tag> {nohead | head | backhead | heads}
                {size <headlength>,<headangle>{,<backangle>}} {fixed}
                {filled | empty | nofilled | noborder}
                {front | back}
                {linestyle | ls <line_style>}
                {linetype | lt <line_type>}
                {linewidth | lw <line_width>}
                {linecolor | lc <colorspec>}
                {dashtype | dt <dashtype>}

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 **set arrow** command with the appropriate tag and specify the parts of the arrow to be changed.

The position of the first end point of the arrow is always specified by "from". The other end point can be specified using any of three different mechanisms. 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. See **coordinates (p. 32)** for details. A coordinate system specifier does not carry over from the first endpoint description the second.

1) "to <position>" specifies the absolute coordinates of the other end.

2) "rto <position>" specifies an offset to the "from" position. 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.

3) "length <coordinate> angle <angle>" specifies the orientation of the arrow in the plane of the graph. Again any of the coordinate systems can be used to specify the length. The angle is always in degrees.

Other characteristics of the arrow can either be specified as a pre-defined arrow style or by providing them in **set arrow** command. For a detailed explanation of arrow characteristics, see **arrowstyle (p. 196)**.

Examples:

To set an arrow pointing from the origin to (1,2) with user-defined linestyle 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.

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. If you want to autoscale based on a subset of the plots in the figure, you can mark the other ones with the flag **noautoscale**. See **datafile (p. 106)**.

Syntax:

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

where <axes> is either **x**, **y**, **z**, **cb**, **x2**, **y2**, **xy**, or **paxis {n}**. 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.

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 **set xrange (p. 215)** for additional information about ranges.

The behavior of autoscaling remains consistent in parametric mode, (see **set parametric (p. 185)**). 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. **autoscale** in parametric mode, then, controls all ranges (t, u, v, x, y, and z) and allows x, y, and z to be fully autoscaled.

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* applying offsets or autoextending the ranges to a whole number of tics, which can cause unexpected results. To prevent this you can explicitly link the secondary axis range to the primary axis range. See **set link (p. 165)**.

Noextend

```
set autoscale noextend
```

By default autoscaling sets the axis range limits to the nearest tic label position that includes all the plot data. Keywords **fixmin**, **fixmax**, **fix** or **noextend** tell gnuplot to disable extension of the axis range to the next tic mark position. In this case the axis range limit exactly matches the coordinate of the most extreme data point. **set autoscale noextend** is a synonym for **set autoscale fix**. Range extension for a single axis can be disabled by appending the **noextend** keyword to the corresponding range command, e.g.

```
set yrange [0:*] noextend
```

set autoscale keepfix autoscales all axes while leaving the fix settings unchanged.

Examples

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
```

Polar mode

When in polar mode (**set polar**), the xrange and the yrange may be left in autoscale mode. If **set rrange** is used to limit the extent of the polar axis, then xrange and yrange will adjust to match this automatically. However, explicit xrange and yrange commands can later be used to make further adjustments. See **set rrange (p. 194)**.

See also `polar demos.`

Bind

show bind shows the current state of all hotkey bindings. See **bind (p. 52)**.

Bmargin

The command **set bmargin** sets the size of the bottom margin. Please see **set margin (p. 167)** for details.

Border

The **set border** and **unset border** 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 {<integer>}
           {front | back | behind}
           {linestyle | ls <line_style>}
           {linetype | lt <line_type>} {linewidth | lw <line_width>}
           {linecolor | lc <colorspec>} {dashtype | dt <dashtype>}
           {polar}
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 four low bits control the border for **plot** and the sides of the base for **splot**; the next four bits control the verticals in **splot**; the four high bits control the edges on top of an **splot**. The border settings is thus the sum of the appropriate entries from the following table:

Graph Border Encoding		
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
4096	polar	no effect

The default setting 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**.

In hidden3d plots the lines making up the border are normally subject to the same hidden3d processing as the plot elements. **set border behind** will override this default.

Using the optional <linestyle>, <linetype>, <linewidth>, <linecolor>, and <dashtype> specifiers, the way the border lines are drawn can be influenced (limited by what the current terminal driver supports). Besides the border itself, this line style is used for the tics, independent of whether they are plotted on the border or on the axes (see **set xtics (p. 217)**).

For **plot**, tics may be drawn on edges other than bottom and left by enabling the second axes – see **set xtics (p. 217)** 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.

The **polar** keyword enables a circular border for polar plots.

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
```

Boxwidth

The **set boxwidth** command is used to set the default width of boxes in the **boxes**, **boxerrorbars**, **boxplot**, **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 **set boxwidth** 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 (p. 166)**) 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 (p. 62)** and **style boxerrorbars (p. 61)** 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 **using** 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
```

Boxdepth

The **set boxdepth** command affects only 3D plots created by **splot with boxes**. It sets the extent of each box along the y axis, i.e. its thickness.

Color

Gnuplot supports two alternative sets of linetypes. The default set uses a different color for each linetype, although it also allows you to draw dotted or dashed lines in that color. The alternative monochrome set uses only dot/dash pattern or linewidth to distinguish linetypes. The **set color** command selects the color linetypes. See **set monochrome (p. 168)**, **set linetype (p. 164)**, and **set colorsequence (p. 135)**.

Colorsequence

Syntax:

```
set colorsequence {default|classic|podo}
```

set colorsequence default selects a terminal-independent repeating sequence of eight colors. See **set linetype (p. 164)**, **colors (p. 48)**.

set colorsequence classic lets each separate terminal type provide its own sequence of line colors. The number provided varies from 4 to more than 100, but most start with red/green/blue/magenta/cyan/yellow. This was the default behaviour prior to version 5.

set colorsequence podo selects eight colors drawn from a set recommended by Wong (2011) [Nature Methods 8:441] as being easily distinguished by color-blind viewers with either protanopia or deuteranopia.

In each case you can further customize the length of the sequence and the colors used. See **set linetype (p. 164)**, **colors (p. 48)**.

Clabel

This command is deprecated. Use **set cntrlabel** instead. **unset clabel** is replaced by **set cntrlabel onecolor**. **set clabel "format"** is replaced by **set cntrlabel format "format"**.

Clip

Syntax:

```
set clip {points|one|two|radial}
unset clip {points|one|two|radial}
show clip
```

Default state:

```
unset clip points
set clip one
unset clip two
unset clip radial
```

Data points whose center lies inside the plot boundaries are normally drawn even if the finite size of the point symbol causes it to extend past a boundary line. **set clip points** causes such points to be clipped (i.e. not drawn) even though the point center is inside the boundaries of a 2D plot. Data points whose center lies outside the plot boundaries are never drawn.

unset clip causes a line segment in a plot not to be drawn if either end of that segment lies outside the plot boundaries (i.e. xrange and yrange).

set clip one causes **gnuplot** to draw the in-range portion of line segments with one endpoint in range and one endpoint out of range. **set clip two** causes **gnuplot** to draw the in-range portion of line segments with both endpoints out of range. Line segments that lie entirely outside the plot boundaries are never drawn.

set clip radial affects plotting only in polar mode. It clips lines against the radial bound established by **set rrange [0:MAX]**. This criteria is applied in conjunction with **set clip {one|two}**. I.e. the portion of a line between two points with R > RMAX that passes through the circle R = RMAX is drawn only if both **clip two** and **clip radial** are set.

Notes:

* **set clip** affects only points and lines produced by plot styles **lines**, **linespoints**, **points**, **arrows**, and **vectors**.

* Clipping of colored quadrangles drawn for pm3d surfaces and other solid objects is controlled **set pm3d clipping**. The default is smooth clipping against the current zrange.

* Object clipping is controlled by the **clip** or **noclip** property of the individual object.

* In the current version of gnuplot, "plot with vectors" in polar mode does not test or clip against the maximum radius.

Cntrlabel

Syntax:

```
set cntrlabel {format "format"} {font "font"}
set cntrlabel {start <int>} {interval <int>}
set contrlabel onecolor
```

set cntrlabel controls the labeling of contours, either in the key (default) or on the plot itself in the case of **splot ... with labels**. In the latter case labels are placed along each contour line according to the **pointinterval** or **pointnumber** property of the label descriptor. By default a label is placed on the 5th line segment making up the contour line and repeated every 20th segment. These defaults are equivalent to

```
set cntrlabel start 5 interval 20
```

They can be changed either via the **set cntrlabel** command or by specifying the interval in the **splot** command itself

```
set contours; splot $FOO with labels point pointinterval -1
```

Setting the interval to a negative value means that the label appear only once per contour line. However if **set samples** or **set isosamples** is large then many contour lines may be created, each with a single label.

A contour label is placed in the plot key for each linetype used. By default each contour level is given its own linetype, so a separate label appears for each. The command **set cntrlabel onecolor** causes all contours to be drawn using the same linetype, so only one label appears in the plot key. This command replaces an older command **unset clabel**.

Cntrparam

set cntrparam controls the generation of contours and their smoothness for a contour plot. **show contour** displays current settings of **cntrparam** as well as **contour**.

Syntax:

```
set cntrparam { { linear
                | cubicspline
                | bspline
                | points <n>
                | order <n>
                | levels { <n>
                         | auto {<n>}
                         | discrete <z1> {,<z2>{,<z3>...}}
                         | incremental <start>, <incr> {,<end>}
                       }
                  {{un}sorted}
                  {firstlinetype N}
                }
              }
show contour
```

This command has two functions. First, it sets the values of z for which contours are to be determined. The number of contour levels <n> should be an integral constant expression. <z1>, <z2> ... are real-valued expressions. Second, it controls the appearance of the individual contour lines.

Keywords controlling the smoothness of contour lines:

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.

Keywords controlling the selection of contour levels:

levels auto — This is the default. <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).

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.

levels 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 multiplicative factor, as it is for **set ztics**, and <end> should not be used.

Keywords controlling the assignment of linetype to contours:

By default the contours are generated in the reverse order specified (**unsorted**). Thus **set cntrparam levels increment 0, 10, 100** will create 11 contours levels starting with 100 and ending with 0. Adding the keyword **sorted** re-orders the contours by increasing numerical value, which in this case would mean the first contour is drawn at 0.

By default contours are drawn using successive linetypes starting with the next linetype after that used for the corresponding surface. Thus **splot x*y lt 5** would use lt 6 for the first contour generated. If **hidden3d** mode is active then each surface uses two linetypes. In this case using default settings would cause the first contour to use the same linetype as the hidden surface, which is undesirable. This can be avoided in either of two ways. (1) Use **set hidden3d offset N** to change the linetype used for the hidden surface. A good choice would be **offset -1** since that will avoid all the contour linetypes. (2) Use the **set cntrparam firstlinetype N** option to specify a block of linetypes used for contour lines independent of whatever was used for the surface. This is particularly useful if you want to customize the set of contour linetypes. N <= 0 restores the default.

If the command **set cntrparam** is given without any arguments specified all options are reset to the default:

```
set cntrparam order 4 points 5
set cntrparam levels auto 5 unsorted
set cntrparam firstlinetype 0
```

Examples

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
```

To define and use a customized block of contour linetypes

```
set linetype 100 lc "red" dt '....'
do for [L=101:199] {
    if (L%10 == 0) {
        set linetype L lc "black" dt solid lw 2
    } else {
        set linetype L lc "gray" dt solid lw 1
    }
}
set cntrparam firstlinetype 100
set cntrparam sorted levels incremental 0, 1, 100
```

See also **set contour (p. 140)** for control of where the contours are drawn, and **set cntrlabel (p. 136)** 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).`

Color box

The color scheme, i.e. the gradient of the smooth color with min_z and max_z values of **pm3d**'s **palette**, is drawn in a color box unless **unset colorbox**.

```
set colorbox
set colorbox {
           { vertical | horizontal } {{no}invert}
           { default | user }
           { origin x, y }
           { size x, y }
           { front | back }
           { noborder | bdefault | border [line style] }
         }
show colorbox
unset colorbox
```

Color box position can be **default** or **user**. If the latter is specified the values as given with the **origin** and **size** subcommands are used. The box can be drawn after (**front**) or before (**back**) the graph or the surface.

The orientation of the color gradient can be switched by options **vertical** and **horizontal**.

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.

border turns the border on (this is the default). **noborder** turns the border off. If an positive integer argument is given after **border**, 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 **cbrange**, **[m]cbtics**, **format cb**, **grid [m]cb**, **cblabel**, and perhaps even **cbdata**, **[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 **set pm3d (p. 187)**, **set palette (p. 180)**, **x11 pm3d (p. 300)**, and **set style line (p. 200)**.

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 color to a particular linetype or linestyle. To see the list of known color names, use the command **show colornames (p. 139)**. Example:

```
set style line 1 linecolor "sea-green"
```

Contour

set contour enables contour drawing for surfaces. This option is available for **splot** only. It requires grid data, see **grid_data (p. 229)** for more details. If contours are desired from non-grid data, **set dgrid3d** can be used to create an appropriate grid.

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, **surface** 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 **set cntrparam (p. 137)** for the parameters that affect the drawing of contours, and **set cntrlabel (p. 136)** for control of labeling of the contours.

The surface can be switched off (see **unset surface (p. 204)**), giving a contour-only graph. Though it is possible to use **set size** to enlarge the plot to fill the screen, more control over the output format can be obtained by writing the contour information to a datablock, and rereading it as a 2D datafile plot:

```
unset surface
set contour
set cntrparam ...
set table $datablock
splot ...
unset table
# contour info now in $datablock
set term <whatever>
plot $datablock
```

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.

While **set contour** is in effect, **splot with <style>** will place the style elements (points, lines, impulses, labels, etc) along the contour lines. **with pm3d** will produce a pm3d surface and also contour lines. If you want to mix other plot elements, say labels read from a file, with the contours generated while **set contour** is active you must append the keyword **nocontours** after that clause in the splot command.

See also **splot datafile (p. 226)**.

See also `contours demo (contours.dem)`

and `contours with user defined levels demo (discrete.dem).`

Dashtype

The **set dashtype** command allows you to define a dash pattern that can then be referred to by its index. This is purely a convenience, as anywhere that would accept the dashtype by its numerical index would also accept an explicit dash pattern. Example:

```
set dashtype 5 (2,4,2,6)   # define or redefine dashtype number 5
plot f1(x) dt 5            # plot using the new dashtype
plot f1(x) dt (2,4,2,6)    # exactly the same plot as above
set linetype 5 dt 5        # always use this dash pattern with linetype 5
set dashtype 66 "..-"      # define a new dashtype using a string
```

See also **dashtype (p. 50)**.

Data style

This form of the command is deprecated. Please see **set style data (p. 198)**.

Datafile

The **set datafile** command options control interpretation of fields read from input data files by the **plot**, **splot**, and **fit** commands. Several options are currently implemented.

Set datafile columnheaders

The **set datafile columnheaders** command guarantees that the first row of input will be interpreted as column headers rather than as data values. It affects all input data sources to plot, splot, fit, and stats commands. If this setting is disabled by **unset datafile columnheaders**, the same effect is triggered on a per-file basis if there is an explicit columnheader() function in a using specifier or plot title associated with that file. See also **set key autotitle (p. 159)** and **columnheader (p. 123)**.

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**.

Set datafile nofpe_trap

The **set datafile nofpe_trap** command tells gnuplot not to re-initialize a floating point exception handler before every expression evaluation used while reading data from an input file. This can significantly speed data input from very large files at the risk of program termination if a floating-point exception is generated.

Set datafile missing

Syntax:

```
set datafile missing "<string>"
set datafile missing NaN
show datafile missing
unset datafile
```

The **set datafile missing** command tells **gnuplot** there is a special string used in input data files to denote a missing data entry. There is no default character for **missing**. Gnuplot makes a distinction between missing data and invalid data (e.g. "NaN", 1/0.). For example invalid data causes a gap in a line drawn through sequential data points; missing data does not.

Non-numeric characters found in a numeric field will usually be interpreted as invalid rather than as a missing data point unless they happen to match the **missing** string.

Conversely **set datafile missing NaN** causes all data or expressions evaluating to not-a-number (NaN) to be treated as missing data.

The example below shows differences between gnuplot version 4 and version 5. Example:

```
set style data linespoints
plot '-' title "(a)"
   1 10
   2 20
```

```
   3 ?
   4 40
   5 50
   e
set datafile missing "?"
plot '-' title "(b)"
   1 10
   2 20
   3 ?
   4 40
   5 50
   e
plot '-' using 1:2 title "(c)"
   1 10
   2 20
   3 NaN
   4 40
   5 50
   e
plot '-' using 1:($2) title "(d)"
   1 10
   2 20
   3 NaN
   4 40
   5 50
   e
```

Plot (a) differs in gnuplot 4 and gnuplot 5 because the third line contains only one valid number. Version 4 switched to a single-datum-on-a-line convention that the line number is "x" and the datum is "y", erroneously placing the point at(2,3).

Both the old and new gnuplot versions handle the same data correctly if the '?' character is designated as a marker for missing data (b).

Old gnuplot versions handled NaN differently depending of the form of the **using** clause, as shown in plots (c) and (d). Gnuplot now handles NaN the same whether the input column was specified as N or ($N). See also the `imageNaN demo.`

Similarly gnuplot 5.4 will notice the missing value flag in column N whether the plot command specifies **using N** or **using ($N)** or **using (func($N))**. However if the "missing" value is encountered during evaluation of some more complicated expression, e.g. **using (column(strcol(1))**, it may evaluate to NaN and be treated as invalid data rather than as a missing data point. If you nevertheless want to treat this as missing data, use the command **set datafile missing NaN**.

Set datafile separator

The command **set datafile separator** tells **gnuplot** that data fields in subsequent input files are separated by a specific character 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 {whitespace | tab | comma | "<chars>"}
```

Examples:

```
# Input file contains tab-separated fields
set datafile separator "\t"
```

```
# Input file contains comma-separated values fields
set datafile separator comma

# Input file contains fields separated by either * or |
set datafile separator "*|"
```

Set datafile commentschars

The command **set datafile commentschars** specifies what characters can be used in a data file to begin comment lines. If the first non-blank character on a line is one of these characters then the rest of the data line is ignored. Default value of the string is "#!" on VMS and "#" otherwise.

Syntax:

```
set datafile commentschars {"<string>"}
show datafile commentschars
unset commentschars
```

Then, the following line in a data file is completely ignored

```
# 1 2 3 4
```

but the following

```
1 # 3 4
```

will be interpreted as garbage in the 2nd column followed by valid data in the 3rd and 4th columns.

Example:

```
set datafile commentschars "#!%"
```

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 matrix (p. 227)** and **binary general (p. 103)** for details about the keywords that can be present in <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=(512,512) format="%uchar"

show datafile binary   # list current settings
```

Decimalsign

The **set decimalsign** command selects a decimal sign for numbers printed into tic labels or **set label** 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, including 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. On linux systems you can get a list of the locales installed on your machine by typing "locale -a". A typical linux locale string is of the form "sl_SI.UTF-8". A typical Windows locale string is of the form "Slovenian_Slovenia.1250" or "slovenian". Please note that interpretation of the locale settings is done by the C library at runtime. Older C libraries may offer only partial support for locale settings such as the thousands grouping separator character.

```
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.

Dgrid3d

The **set dgrid3d** command enables, and can set parameters for, non-grid to grid data mapping. See **splot grid_data (p. 229)** for more details about the grid data structure.

Syntax:

```
set dgrid3d {<rows>} {,{<cols>}}
            { splines |
              qnorm {<norm>} |
              (gauss | cauchy | exp | box | hann)
                {kdensity} {<dx>} {,<dy>} }
unset dgrid3d
show dgrid3d
```

By default **dgrid3d** is disabled. When enabled, 3D 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 or spline interpolations of the scattered points' z values. In other words, a regularly spaced grid is created and the a smooth approximation to the raw data is evaluated for all grid points. This approximation is plotted in place of the raw data.

The number of columns defaults to the number of rows, which defaults to 10.

Several algorithms are available to calculate the approximation from the raw data. Some of these algorithms can take additional parameters. These interpolations are such the closer the data point is to a grid point, the more effect it has on that grid point.

The **splines** algorithm calculates an interpolation based on "thin plate splines". It does not take additional parameters.

The **qnorm** algorithm calculates a weighted average of the input data at each grid point. Each data point is weighted by the inverse of its distance from the grid point raised to some power. The power is specified as an optional integer parameter that defaults to 1. This algorithm is the default.

Finally, several smoothing kernels are available to calculate weighted averages: z = Sum_i w(d_i) * z_i / Sum_i w(d_i), where z_i is the value of the i-th data point and d_i is the distance between the current grid point and the location of the i-th data point. All kernels assign higher weights to data points that are close to the current grid point and lower weights to data points further away.

The following kernels are available:

```
gauss :     w(d) = exp(-d*d)
cauchy :    w(d) = 1/(1 + d*d)
exp :       w(d) = exp(-d)
box :       w(d) = 1                  if d<1
                 = 0                  otherwise
hann :      w(d) = 0.5*(1+cos(pi*d))  if d<1
            w(d) = 0                  otherwise
```

When using one of these five smoothing kernels, up to two additional numerical parameters can be specified: dx and dy. These are used to rescale the coordinate differences when calculating the distance: d_i = sqrt(((x-x_i)/dx)**2 + ((y-y_i)/dy)**2), where x,y are the coordinates of the current grid point and x_i,y_i are the coordinates of the i-th data point. The value of dy defaults to the value of dx, which defaults to 1. The parameters dx and dy make it possible to control the radius over which data points contribute to a grid point IN THE UNITS OF THE DATA ITSELF.

The optional keyword **kdensity**, which must come after the name of the kernel, but before the (optional) scale parameters, modifies the algorithm so that the values calculated for the grid points are not divided by the sum of the weights (z = Sum_i w(d_i) * z_i). If all z_i are constant, this effectively plots a bivariate kernel density estimate: a kernel function (one of the five defined above) is placed at each data point, the sum of these kernels is evaluated at every grid point, and this smooth surface is plotted instead of the original data. This is similar in principle to + what the **smooth kdensity** option does to 1D datasets. (See kdensity2d.dem for usage demo)

A slightly different syntax is also supported for reasons of backwards compatibility. If no interpolation algorithm has been explicitly selected, the **qnorm** algorithm is assumed. Up to three comma-separated, optional parameters can be specified, which are interpreted as the the number of rows, the number of columns, and the norm value, respectively.

The **dgrid3d** option is a simple scheme which replaces scattered data with weighted averages on a regular grid. More sophisticated approaches to this problem exist and should be used to preprocess the data outside **gnuplot** if this simple solution is found inadequate.

See also `dgrid3d.dem: dgrid3d demo.`

and `scatter.dem: dgrid3d demo.`

Dummy

The **set dummy** 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)
```

Examples:

```
set dummy u,v
set dummy ,s
```

The second example sets the second variable to s. To reset the dummy variable names to their default values, use

```
unset dummy
```

Encoding

The **set encoding** command selects a character encoding.

Syntax:

```
set encoding {<value>}
set encoding locale
show encoding
```

Valid values are

```
default     - tells a terminal to use its default encoding
iso_8859_1  - the most common Western European encoding prior to UTF-8.
              Known in the PostScript world as 'ISO-Latin1'.
iso_8859_15 - a variant of iso_8859_1 that includes the Euro symbol
iso_8859_2  - used in Central and Eastern Europe
iso_8859_9  - used in Turkey (also known as Latin5)
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
cp950       - MS version of Big5 (emf terminal only)
cp1250      - codepage for MS Windows, Central and Eastern Europe
cp1251      - codepage for 8-bit Russian, Serbian, Bulgarian, Macedonian
cp1252      - codepage for MS Windows, Western Europe
cp1254      - codepage for MS Windows, Turkish (superset of Latin5)
sjis        - shift-JIS Japanese encoding
utf8        - variable-length (multibyte) representation of Unicode
              entry point for each character
```

The command **set encoding locale** is different from the other options. It attempts to determine the current locale from the runtime environment. On most systems this is controlled by the environmental variables LC_ALL, LC_CTYPE, or LANG. This mechanism is necessary, for example, to pass multibyte character encodings such as UTF-8 or EUC_JP to the wxt and pdf terminals. This command does not affect the locale-specific representation of dates or numbers. See also **set locale (p. 166)** and **set decimalsign (p. 143)**.

Generally you must set the encoding before setting the terminal type, as it may affect the choice of appropriate fonts.

Errorbars

The **set errorbars** command controls the tics at the ends of error bars, and also at the end of the whiskers belonging to a boxplot.

Syntax:

```
set errorbars {small | large | fullwidth | <size>} {front | back}
              {line-properties}
unset errorbars
show errorbars
```

small is a synonym for 0.0 (no crossbar), and **large** for 1.0. The default is 1.0 if no size is given.

The keyword **fullwidth** is relevant only to boxplots and to histograms with errorbars. It sets the width of the errorbar ends to be the same as the width of the associated box. It does not change the width of the box itself.

The **front** and **back** keywords are relevant only to errorbars attached to filled rectangles (boxes, candlesticks, histograms).

Error bars are by default drawn using the same line properties as the border of the associated box. You can change this by providing a separate set of line properties for the error bars.

```
set errorbars linecolor black linewidth 0.5 dashtype '.'
```

Fit

The **set fit** command controls the options for the **fit** command.

Syntax:

```
set fit {nolog | logfile {"<filename>"|default}}
        {{no}quiet|results|brief|verbose}
        {{no}errorvariables}
        {{no}covariancevariables}
        {{no}errorscaling}
        {{no}prescale}
        {maxiter <value>|default}
        {limit <epsilon>|default}
        {limit_abs <epsilon_abs>}
        {start-lambda <value>|default}
        {lambda-factor <value>|default}
        {script {"<command>"|default}}
        {v4 | v5}
unset fit
show fit
```

The **logfile** option defines where the **fit** command writes its output. 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**. 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.

By default the information written to the log file is also echoed to the terminal session. **set fit quiet** turns off the echo, whereas **results** prints only final results. **brief** gives one line summaries for every iteration of the fit in addition. **verbose** yields detailed iteration reports as in version 4.

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 1 sprintf("a=%6.2f +/- %6.2f", a, a_err)
plot 'datafile' using 1:2, f(x)
```

If the **errorscaling** option is specified, which is the default, the calculated parameter errors are scaled with the reduced chi square. This is equivalent to providing data errors equal to the calculated standard deviation of the fit (FIT_STDFIT) resulting in a reduced chi square of one. With the **noerrorscaling** option the estimated errors are the unscaled standard deviations of the fit parameters. If no weights are specified for the data, parameter errors are always scaled.

If the **prescale** option is turned on, parameters are prescaled by their initial values before being passed to the Marquardt-Levenberg routine. This helps tremendously if there are parameters that differ in size by many orders of magnitude. Fit parameters with an initial value of exactly zero are never prescaled.

The maximum number of iterations may be limited with the **maxiter** option. A value of 0 or **default** means that there is no limit.

The **limit** option can be used to change the default epsilon limit (1e-5) to detect convergence. When the sum of squared residuals changes by a factor less than this number (epsilon), the fit is considered to have 'converged'. The **limit_abs** option imposes an additional absolute limit in the change of the sum of squared residuals and defaults to zero.

If you need even more control about the algorithm, and know the Marquardt-Levenberg algorithm well, the following options can be used 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 using the **start_lambda** option. Setting it to **default** will re-enable the automatic selection. The option **lambda_factor** sets the factor by which **lambda** is increased or decreased whenever the chi-squared target function increased or decreased significantly. Setting it to **default** re-enables the default factor of 10.0.

The **script** option may be used to specify a **gnuplot** command to be executed when a fit is interrupted — see **fit (p. 90)**. This setting takes precedence over the default of **replot** and the environment variable **FIT_SCRIPT**.

If the **covariancevariables** option is turned on, the covariances between final parameters will be saved to user-defined variables. The variable name for a certain parameter combination is formed by prepending "FIT_COV_" to the name of the first parameter and combining the two parameter names by "_". For example given the parameters "a" and "b" the covariance variable is named "FIT_COV_a_b".

In version 5 the syntax of the fit command changed and it now defaults to unitweights if no 'error' keyword is given. The **v4** option restores the default behavior of gnuplot version 4, see also **fit (p. 90)**.

Fontpath

Syntax:

```
set fontpath "/directory/where/my/fonts/live"
set term postscript fontfile <filename>
```

[DEPRECATED in version 5.4]

The **fontpath** directory is relevant only for embedding fonts in postscript output produced by the postscript terminal. It has no effect on other gnuplot terminals. If you are not embedding fonts you do not need this command, and even if you are embedding fonts you only need it for fonts that cannot be found via the other paths below.

Earlier versions of gnuplot tried to emulate a font manager by tracking multiple directory trees containing fonts. This is now replaced by a search in the following places: (1) an absolute path given in the **set term postscript fontfile** command (2) the current directory (3) any of the directories specified by **set**

loadpath (4) the directory specified by **set fontpath** (5) the directory provided in environmental variable GNUPLOT_FONTPATH

Note: The search path for fonts specified by filename for the libgd terminals (png gif jpeg sixel) is controlled by environmental variable GDFONTPATH.

Format

The format of the tic-mark labels can be set with the **set format** command or with the **set tics format** or individual **set {axis}tics format** commands.

Syntax:

```
set format {<axes>} {"<format-string>"} {numeric|timedate|geographic}
show format
```

where <axes> is either **x**, **y**, **xy**, **x2**, **y2**, **z**, **cb** or nothing (which applies the format to all axes). The following two commands are equivalent:

```
set format y "%.2f"
set ytics format "%.2f"
```

The length of the string is restricted to 100 characters. The default format is "% h", "$%h$" for LaTeX terminals. Other formats such as "%.2f" or "%3.0em" are often desirable. "set format" with no following string will restore the default.

If the empty string "" is given, tics will have no labels, although the tic mark will still be plotted. To eliminate the tic marks, use **unset xtics** or **set tics scale 0**.

Newline (\n) and enhanced text markup is accepted in the format string. Use double-quotes rather than single-quotes in this case. See also **syntax (p. 59)**. 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 **set xtics (p. 217)** for more information about tic labels, and **set decimalsign (p. 143)** for how to use non-default decimal separators in numbers printed this way. See also `electron demo (electron.dem).`

Gprintf

The string function gprintf("format",x) uses gnuplot's own format specifiers, as do the gnuplot commands **set format**, **set timestamp**, and others. These format specifiers are not the same as those used by the standard C-language routine sprintf(). gprintf() accepts only a single variable to be formatted. Gnuplot also provides an sprintf("format",x1,x2,...) routine if you prefer. For a list of gnuplot's format options, see **format specifiers (p. 149)**.

Format specifiers

The acceptable formats (if not in time/date mode) are:

Tic-mark label numerical format specifiers	
Format	Explanation
`%f`	floating point notation
`%e` or `%E`	exponential notation; an "e" or "E" before the power
`%g` or `%G`	the shorter of `%e` (or `%E`) and `%f`
`%h` or `%H`	like `%g with "x10^{%S}" or "*10^{%S}" instead of "e%S"`
`%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; scientific power
`%T`	power to base 10
`%L`	power to base of current logscale
`%S`	scientific power
`%c`	character replacement for scientific power
`%b`	mantissa of ISO/IEC 80000 notation (ki, Mi, Gi, Ti, Pi, Ei, Zi, Yi)
`%B`	prefix of ISO/IEC 80000 notation (ki, Mi, Gi, Ti, Pi, Ei, Zi, Yi)
`%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 **set timefmt (p. 208)** for a list of the allowed input format codes.

Time/date specifiers

There are two groups of time format specifiers: time/date and relative time. These may be used to generate axis tic labels or to encode time in a string. See **set xtics time (p. 220)**, **strftime (p. 39)**, **strptime (p. 39)**.

The time/date formats are

Date Specifiers	
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, 01–31
`%D`	shorthand for `"%m/%d/%y"` (only output)
`%F`	shorthand for `"%Y-%m-%d"` (only output)
`%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, 001–366
`%m`	month, 01–12
`%M`	minute, 00–60
`%p`	"am" or "pm"
`%r`	shorthand for `"%I:%M:%S %p"` (only output)
`%R`	shorthand for `%H:%M"` (only output)
`%S`	second, integer 00–60 on output, (double) on input
`%s`	number of seconds since start of year 1970
`%T`	shorthand for `"%H:%M:%S"` (only output)
`%U`	week of the year (CDC/MMWR "epi week") (ignored on input)
`%w`	day of the week, 0–6 (Sunday = 0)
`%W`	week of the year (ISO 8601 week date) (ignored on input)
`%y`	year, 0-99 in range 1969-2068
`%Y`	year, 4-digit
`%z`	timezone, [+-]hh:mm
`%Z`	timezone name, ignored string

For more information on the %W format (ISO week of year) see **tm_week (p. 152)**. The %U format (CDC/MMWR epidemiological week) is similar to %W except that it uses weeks that start on Sunday rather than Monday. Caveat: Both the %W and the %U formats were unreliable in gnuplot versions prior to 5.4.2. See unit test `"week_date.dem"`.

The relative time formats express the length of a time interval on either side of a zero time point. The relative time formats are

Time Specifiers	
Format	Explanation
`%tD`	+/- days relative to time=0
`%tH`	+/- hours relative to time=0 (does not wrap at 24)
`%tM`	+/- minutes relative to time=0
`%tS`	+/- seconds associated with previous tH or tM field

Numerical formats may be preceded by a `"0"` (`"zero"`) to pad the field with leading zeroes, and preceded by a positive digit to define the minimum field width. The %S, and %t formats also accept a precision specifier so that fractional hours/minutes/seconds can be written.

Examples Examples of date format:

Suppose the x value in seconds corresponds a time slightly before midnight on 25 Dec 1976. The text printed for a tic label at this position would be

```
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"
```

Examples of time format:

The date format specifiers encode a time in seconds as a clock time on a particular day. So hours run only from 0-23, minutes from 0-59, and negative values correspond to dates prior to the epoch (1-Jan-1970). In order to report a time value in seconds as some number of hours/minutes/seconds relative to a time 0, use time formats %tH %tM %tS. To report a value of -3672.50 seconds

```
set format x                 # default date format "12/31/69 \n 22:58"
set format x "%tH:%tM:%tS"   # "-01:01:12"
set format x "%.2tH hours"   # "-1.02 hours"
set format x "%tM:%.2tS"     # "-61:12.50"
```

Tm_week

The **tm_week(t, standard)** function interprets its first argument t as a time in seconds from 1 Jan 1970. Despite the name of this function it does not report a field from the POSIX tm structure.

If standard = 0 it returns the week number in the ISO 8601 "week date" system. This corresponds to gnuplot's %W time format. If standard = 1 it returns the CDC epidemiological week number ("epi week"). This corresponds to gnuplot's %U time format. For corresponding inverse functions that convert week dates to calendar time see **weekdate_iso (p. 152)**, **weekdate_cdc (p. 153)**.

In brief, ISO Week 1 of year YYYY begins on the Monday closest to 1 Jan YYYY. This may place it in the previous calendar year. For example Tue 30 Dec 2008 has ISO week date 2009-W01-2 (2nd day of week 1 of 2009). Up to three days at the start of January may come before the Monday of ISO week 1; these days are assigned to the final week of the previous calendar year. E.g. Fri 1 Jan 2021 has ISO week date 2020-W53-5.

The US Center for Disease Control (CDC) epidemiological week is a similar week date convention that differs from the ISO standard by defining a week as starting on Sunday, rather than on Monday.

Weekdate_iso

Syntax:

```
time = weekdate_iso( year, week [, day] )
```

This function converts from the year, week, day components of a date in ISO 8601 "week date" format to the calendar date as a time in seconds since the epoch date 1 Jan 1970. Note that the nominal year in the week date system is not necessarily the same as the calendar year. The week is an integer from 1 to 53. The day parameter is optional. If it is omitted or equal to 0 the time returned is the start of the week. Otherwise day is an integer from 1 (Monday) to 7 (Sunday). See **tm_week (p. 152)** for additional information on an inverse function that converts from calendar date to week number in the ISO standard convention.

Example:

```
# Plot data from a file with column 1 containing ISO weeks
#     Week     cases  deaths
#     2020-05    432       1
calendar_date(w) = weekdate_iso( int(w[1:4]), int(w[6:7]) )
set xtics time format "%b\n%Y"
plot FILE using (calendar_date(strcol(1))) : 2   title columnhead
```

Weekdate_cdc

Syntax:
```
time = weekdate_cdc( year, week [, day] )
```

This function converts from the year, week, day components of a date in the CDC/MMWR "epi week" format to the calendar date as a time in seconds since the epoch date 1 Jan 1970. The CDC week date convention differs from the ISO week date in that it is defined in terms of each week running from day 1 = Sunday to day 7 = Saturday. If the third parameter is 0 or is omitted, the time returned is the start of the week. See **tm_week (p. 152)** and **weekdate_iso (p. 152)**.

Function style

This form of the command is deprecated. Please use **set style function**.

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 (p. 36)**. See also `splines as user defined functions (spline.dem)`

and `use of functions and complex variables for airfoils (airfoil.dem).`

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}rtics}
         {{no}{m}cbtics}
         {polar {<angle>}}
         {layerdefault | front | back}
         {{no}vertical}
         {<line-properties-major> {, <line-properties-minor>}}
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 (see **set style line (p. 200)**).

A polar grid can be drawn for 2D plots. This is the default action of **set grid** if the program is already in polar mode, but can be enabled explicitly by **set grid polar <angle> rtics** whether or not the program is in polar mode. Circles are drawn to intersect major and/or minor tics along the r axis, and radial lines are drawn with a spacing of <angle>. Tic marks around the perimeter are controlled by **set ttics**, but these do not produce radial grid lines.

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 **hidden3d** 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 **set border** and the various ticmarks (see **set xtics (p. 217)**).

In 3D plots grid lines at x- and y- axis tic positions are by default drawn only on the base plane parallel to z=0. The **vertical** keyword activates drawing grid lines in the xz and yz planes also, running from zmin to zmax.

Z grid lines are drawn on the bottom of the plot. This looks better if a partial box is drawn around the plot — see **set border (p. 133)**.

Hidden3d

The **set hidden3d** command enables hidden line removal for surface plotting (see **splot (p. 225)**). Some optional features of the underlying algorithm can also be controlled using this command.

Syntax:

```
set hidden3d {defaults} |
             { {front|back}
               {{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 plot elements behind the surface will be hidden by it. For this to work, the surface needs to have 'grid structure' (see **splot datafile (p. 226)** about this), and it has to be drawn **with lines** or **with linespoints**.

When **hidden3d** is set, both the hidden portion of the surface and possibly its contours drawn on the base (see **set contour (p. 140)**) 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 (**set contour surface**) don't work.

hidden3d also affects 3D plotting styles **points**, **labels**, **vectors**, and **impulses** even if no surface is present in the graph. Unobscured portions of each vector are drawn as line segments (no arrowheads). Individual plots within the graph may be explicitly excluded from this processing by appending the extra option **nohidden3d** to the **with** specifier.

Hidden3d does not affect solid surfaces drawn using the pm3d mode. To achieve a similar effect purely for pm3d surfaces, use instead **set pm3d depthorder**. To mix pm3d surfaces with normal **hidden3d** processing, use the option **set hidden3d front** to force all elements included in hidden3d processing to be drawn after any remaining plot elements, including the pm3d surface.

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 **hidden3d** than when plotted with **nohidden3d** because in the latter case functions are evaluated at each sample. Please see **set samples (p. 194)** and **set isosamples (p. 156)** 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 linetype used for lines on the 'back' side. Normally, they are drawn in a linetype 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 linetype 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 linetype.

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).`

Historysize

(Deprecated). **set historysize N** is equivalent to **set history size N**. **unset historysize** is equivalent to **set history size -1**.

History

Syntax:

```
set history {size <N>} {quiet|numbers} {full|trim} {default}
```

When leaving gnuplot the value of history size limits the number of lines saved to the history file. **set history size -1** allows an unlimited number of lines to be written to the history file.

By default the **history** command prints a line number in front of each command. **history quiet** suppresses the number for this command only. **set history quiet** suppresses numbers for all future **history** commands.

The **trim** option reduces the number of duplicate lines in the history list by removing earlier instances of the current command.

Default settings: **set history size 500 numbers trim**.

Isosamples

The isoline density (grid) for plotting functions as surfaces may be changed by the **set isosamples** 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, **set samples** controls the number of points sampled along each isoline; see **set samples (p. 194)** and **set hidden3d (p. 154)**. The contour algorithm assumes that a function sample occurs at each isoline intersection, so change in **samples** as well as **isosamples** may be desired when changing the resolution of a function surface/contour.

Isosurface

Syntax:

```
set isosurface {mixed|triangles}
set isosurface {no}insidecolor <n>
```

Surfaces plotted by the command **splot $voxelgrid with isosurface** are by default constructed from a mixture of quadrangles and triangles. The use of quadrangles creates a less complicated visual impression. This is the default. This command proveds an option to tessellate with only triangles.

By default the inside of an isosurface is drawn in a separate color. The method of choosing that color is the same as for hidden3d surfaces, where an offset <n> is added to the base linetype. To draw both the inside and outside surfaces in the same color, use **set isosurface noinsidecolor**.

Jitter

Syntax:

```
set jitter {overlap <yposition>} {spread <factor>} {wrap <limit>}
           {swarm|square|vertical}
```

Examples:

```
set jitter                    # jitter points within 1 character width
set jitter overlap 1.5        # jitter points within 1.5 character width
set jitter over 1.5 spread 0.5  # same but half the displacement on x
```

When one or both coordinates of a data set are restricted to discrete values then many points may lie exactly on top of each other. Jittering introduces an offset to the coordinates of these superimposed points that spreads them into a cluster. The threshold value for treating the points as being overlapped may be specified in character widths or any of the usual coordinate options. See **coordinates (p. 32)**. Jitter affects 2D plot styles **with points** and **with impulses**. It also affects 3D plotting of voxel grids.

The default jittering operation displaces points only along x. This produces a distinctive pattern sometimes called a "bee swarm plot". The optional keyword **square** adjusts the y coordinate of displaced points in addition to their x coordinate so that the points lie in distinct layers separated by at least the **overlap** distance.

To jitter along y (only) rather than along x, use keyword **vertical**.

The maximum displacement (in character units) can be limited using the **wrap** keyword.

Note that both the overlap criterion and the magnitude of jitter default to one character unit. Thus the plot appearance will change with the terminal font size, canvas size, or zoom factor. To avoid this you can specify the overlap criterion in the y axis coordinate system (the **first** keyword) and adjust the point size and spread multiplier as appropriate. See **coordinates (p. 32)**, **pointsize (p. 192)**.

Caveat: jitter is incompatible with "pointsize variable".

set jitter is also useful in 3D plots of voxel data. Because voxel grids are regular lattices of evenly spaced points, many view angles cause points to overlap and/or generate Moiré patterns. These artifacts can be removed by displacing the symbol drawn at each grid point by a random amount.

Key

The **set key** command enables a key (or legend) containing a title and a sample (line, point, box) for each plot in the graph. The key may be turned off by requesting **set key off** or **unset key**. Individual key entries may be turned off by using the **notitle** keyword in the corresponding plot command. The text of the titles is controlled by the **set key autotitle** option or by the **title** keyword of individual **plot** and **splot** commands. See **plot title (p. 123)** for more information.

Syntax:

```
set key {on|off} {default}
      {{inside | outside | fixed} | {lmargin | rmargin | tmargin | bmargin}
        | {at <position>}}
      {left | right | center} {top | bottom | center}
      {vertical | horizontal} {Left | Right}
      {{no}enhanced}
      {{no}opaque {fc <colorspec>}}
      {{no}reverse} {{no}invert}
      {samplen <sample_length>} {spacing <line_spacing>}
      {width <width_increment>} {height <height_increment>}
      {{no}autotitle {columnheader}}
      {title {"<text>"} {{no}enhanced} {center | left | right}}
      {font "<face>,<size>"} {textcolor <colorspec>}
      {{no}box {linestyle <style> | linetype <type> | linewidth <width>}}
      {maxcols {<max no. of columns> | auto}}
      {maxrows {<max no. of rows> | auto}}
unset key
show key
```

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. The vertical space may be limited using 'maxrows'. In the case of **horizontal**, the key occupies as few rows as possible. The horizontal space may be limited using 'maxcols'.

By default the key is placed in the upper right inside corner of the graph. The keywords **left**, **right**, **top**, **bottom**, **center**, **inside**, **outside**, **lmargin**, **rmargin**, **tmargin**, **bmargin** (, **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**, **top**, **bottom** and **center** serve an analogous purpose for alignment. For more information, see **key placement (p. 160)**.

Justification of the plot titles within the key is controlled by **Left** or **Right** (default). 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**.

The text in the key is set in **enhanced** mode by default, this can be changed with the **{no}enhanced** option, also independently for the key title only and for each individual plot.

By default the key is built up one plot at a time. That is, the key symbol and title are drawn at the same time as the corresponding plot. That means newer plots may sometimes place elements on top of the key. **set key opaque** causes the key to be generated after all the plots. In this case the key area is filled with background color or the requested fill color and then the key symbols and titles are written. The default can be restored by **set key noopaque**.

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 <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 and want larger borders around the key entries.

An overall title can be put on the key (**title "<text>"**) — see also **syntax (p. 59)** for the distinction between text in single- or double-quotes. The justification of the title defaults to center and can be changed by the keywords **right** or **left**

The defaults for **set key** are **on**, **right**, **top**, **vertical**, **Right**, **noreverse**, **noinvert**, **samplen 4**, **spacing 1**, **notitle**, 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.

Each plot is represented in the key by a single line containing a line or symbol or shape representing the plot style and a corresponding title. Using the keyword **notitle** in the plot command will suppress generation of the line. Contour plots generated additional entries in the key, one for each contour (see **cntrlabel (p. 136)**). You can add extra lines to the key by inserting a dummy plot command that uses the keyword **keyentry** rather than a filename or a function. See **keyentry (p. 159)**.

When using the TeX/LaTeX group of terminals or terminals in which formatting information is embedded in the string, **gnuplot** can only estimate 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**.

3D key

Placement of the key for 3D plots (**splot**) by default uses the **fixed** option. Note: this is a change from gnuplot version 5.0 and earlier. **fixed** placement is very similar to **inside** placement with one important difference. The plot boundaries of a 3D plot change as the view point is rotated or scaled. If the key is positioned **inside** these boundaries then the key also moves when the view is changed. **fixed** positioning ignores changes to the view angles or scaling; i.e. the key remains fixed in one location on the canvas as the plot is rotated.

For 2D plots the **fixed** option is exactly equivalent to **inside**.

If **splot** is being used to draw contours, by default a separate key entry is generated for each contour level with a distinct line type. To modify this see **set cntrlabel (p. 136)**.

Key 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 at 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
```

Extra key entries

Normally each plot autogenerates a single line entry in the key. If you need more control over what appears in the key you can use the **keyentry** keyword in the **plot** or **splot** command to insert extra lines. Instead of providing a filename or function to plot, use **keyentry** as a placeholder followed by plot style information (used to generate a key symbol) and a title. All the usual options for title font, text color, **at** coordinates, and enhanced text markup apply. Example:

```
plot $HEATMAP matrix with image notitle, \
     keyentry with boxes fc palette cb 0 title "no effect", \
     keyentry with boxes fc palette cb 1 title "threshold", \
     keyentry with boxes fc palette cb 3 title "typical range", \
     keyentry with labels nopoint title "as reported in [12]", \
     keyentry with boxes fc palette cb 5 title "strong effect"
```

Key autotitle

set key autotitle causes each plot to be identified in the key by the name of the data file or function used in the plot command. This is the default. **set key noautotitle** disables the automatic generation of plot titles. The command **set key autotitle columnheader** causes the first entry in each column of input 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 columnhead(3) with lines
```

Note: The effect of **set key autotitle columnheader**, treatment of the first line in a data file as column headers rather than data applies even if the key is disabled by **unset key**. It also applies to **stats** and **fit** commands even though they generate no key. If you want the first line of data to be treated as column headers but *not* to use them for plot titles, use **set datafile columnheaders**.

In all cases an explicit **title** or **notitle** keyword in the plot command itself will override the default from **set key autotitle**.

Key placement

This section describes placement of the primary, auto-generated key. To construct a secondary key or place plot titles elsewhere, see **multiple keys (p. 161)**.

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 **lmargin** (lm), **rmargin** (rm), **tmargin** (tm), and **bmargin** (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 **tmargin**. 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 **bmargin**. 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 (p. 32)** 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.

Key samples

By default, each plot on the graph generates a corresponding entry in the key. This entry contains a plot title and a sample line/point/box of the same color and fill properties as used in the plot itself. The font and textcolor properties control the appearance of the individual plot titles that appear in the key. Setting the textcolor to "variable" causes the text for each key entry to be the same color as the line or fill color for that plot. This was the default in some earlier versions of gnuplot.

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.

Key entry lines are single-spaced based on the current font size. This can be adjusted by **set key spacing <line-spacing>**.

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.

Multiple keys

It is possible to construct a legend/key manually rather than having the plot titles all appear in the auto-generated key. This allows, for example, creating a single legend for the component panels in a multiplot. Here is an example:

```
set multiplot layout 3,2 columnsfirst
set style data boxes
plot $D using 0:6  lt 1 title at 0.75, 0.20
plot $D using 0:12 lt 2 title at 0.75, 0.17
plot $D using 0:13 lt 3 title at 0.75, 0.14
plot $D using 0:14 lt 4 title at 0.75, 0.11
set label 1 at screen 0.75, screen 0.22 "Custom combined key area"
plot $D using 0:($6+$12+$13+$14) with linespoints title "total"
unset multiplot
```

Label

Arbitrary labels can be placed on the plot using the **set label** 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>}
          {nobox} {boxed {bs <boxstyle>}}
          {hypertext}
unset label {<tag>}
show label
```

The <position> is specified by either x,y or x,y,z, and may be preceded by **first**, **second**, **polar**, **graph**, **screen**, or **character** to indicate the coordinate system. See **coordinates (p. 32)** 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 **set label** 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 (p. 56)**, **sprintf (p. 39)**, and **gprintf (p. 149)**.

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.

Some terminals support enclosing the label in a box. See **set style textbox (p. 203)**. Not all terminals can handle boxes for rotated text.

If **rotate** is given, the label is written vertically. If **rotate by <degrees>** is given, the baseline of the text will be set to the specified angle. Some terminals do not support text rotation.

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 (p. 34)**.

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> can be a linetype, an rgb color, or a palette mapping. See help for **colorspec (p. 49)** and **palette (p. 39)**. **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 label z position.
‘tc palette cb <val>‘ selects a color corresponding to <val> on the colorbar.
‘tc palette fraction <val>‘, with 0<=val<=1, selects a color corresponding to
    the mapping [0:1] to grays/colors of the ‘palette‘.
‘tc rgb "#RRGGBB"‘ or ‘tc rgb "0xRRGGBB"‘ sets an arbitrary 24-bit RGB color.
‘tc rgb 0xRRGGBB‘  As above; a hexadecimal constant does not require quotes.
```

If a <pointstyle> is given, using keywords **lt**, **pt** and **ps**, see **style (p. 124)**, 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 **pointsize** 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 (p. 32)** for details.

If one (or more) axis is timeseries, the appropriate coordinate should be given as a quoted time string according to the **timefmt** format string. See **set xdata (p. 213)** and **set timefmt (p. 208)**.

The options available for **set label** are also available for the **labels** plot style. See **labels (p. 75)**. In this case the properties **textcolor**, **rotate**, and **pointsize** may be followed by keyword **variable** rather than by a fixed value. In this case the corresponding property of individual labels is determined by additional columns in the **using** specifier.

Examples

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 display a function definition along with its fitted parameters, use:

```
f(x)=a+b*x
fit f(x) 'datafile' via a,b
set label GPFUN_f at graph .05,.95
set label sprintf("a = %g", a) at graph .05,.90
set label sprintf("b = %g", b) at graph .05,.85
```

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
```

Hypertext

Some terminals (wxt, qt, svg, canvas, win) allow you to attach hypertext to specific points on the graph or elsewhere on the canvas. When the mouse hovers over the anchor point, a pop-up box containing the text is displayed. Terminals that do not support hypertext will display nothing. You must enable the **point** attribute of the label in order for the hypertext to be anchored. Examples:

```
set label at 0,0 "Plot origin" hypertext point pt 1
plot 'data' using 1:2:0 with labels hypertext point pt 7 \
     title 'mouse over point to see its order in data set'
```

For the wxt and qt terminals, left-click on a hypertext anchor after the text has appeared will copy the hypertext to the clipboard.

EXPERIMENTAL (implementation details may change) Text of the form "image{<xsize>,<ysize>}:<filename>{\n<caption text>}" will trigger display of the image file in a pop-up box. The optional size overrides a default box size 300x200. The types of image file recognized may vary by terminal type, but *.png should always work. Any additional text lines following the image filename are displayed as usual for hypertext. Example:

```
set label 7 "image:../figures/Fig7_inset.png\nFigure 7 caption..."
set label 7 at 10,100 hypertext point pt 7
```

Linetype

The **set linetype** command allows you to redefine the basic linetypes used for plots. The command options are identical to those for "set style line". Unlike line styles, redefinitions by **set linetype** are persistent; they are not affected by **reset**.

For example, whatever linetypes one and two look like to begin with, if you redefine them like this:

```
set linetype 1 lw 2 lc rgb "blue" pointtype 6
set linetype 2 lw 2 lc rgb "forest-green" pointtype 8
```

everywhere that uses lt 1 will now get a thick blue line. This includes uses such as the definition of a temporary linestyle derived from the base linetype 1. Similarly lt 2 will now produce a thick green line.

This mechanism can be used to define a set of personal preferences for the sequence of lines used in gnuplot. The recommended way to do this is to add to the run-time initialization file ~ /.gnuplot a sequence of commands like

```
set linetype 1 lc rgb "dark-violet" lw 2 pt 1
set linetype 2 lc rgb "sea-green"   lw 2 pt 7
set linetype 3 lc rgb "cyan"        lw 2 pt 6 pi -1
set linetype 4 lc rgb "dark-red"    lw 2 pt 5 pi -1
set linetype 5 lc rgb "blue"        lw 2 pt 8
set linetype 6 lc rgb "dark-orange" lw 2 pt 3
set linetype 7 lc rgb "black"       lw 2 pt 11
set linetype 8 lc rgb "goldenrod"   lw 2
set linetype cycle 8
```

Every time you run gnuplot the line types will be initialized to these values. You may initialize as many linetypes as you like. If you do not redefine, say, linetype 3 then it will continue to have the default properties (in this case blue, pt 3, lw 1, etc).

Similar script files can be used to define theme-based color choices, or sets of colors optimized for a particular plot type or output device.

The command **set linetype cycle 8** tells gnuplot to re-use these definitions for the color and linewidth of higher-numbered linetypes. That is, linetypes 9-16, 17-24, and so on will use this same sequence of colors and widths. The point properties (pointtype, pointsize, pointinterval) are not affected by this command. **unset linetype cycle** disables this feature. If the line properties of a higher numbered linetype are explicitly defined, this takes precedence over the recycled low-number linetype properties.

Link

Syntax:

```
set link {x2 | y2} {via <expression1> inverse <expression2>}
unset link
```

The **set link** command establishes a mapping between the x and x2 axes, or the y and y2 axes. <expression1> maps primary axis coordinates onto the secondary axis. <expression2> maps secondary axis coordinates onto the primary axis.

Examples:

```
set link x2
```

This is the simplest form of the command. It forces the x2 axis to have identically the same range, scale, and direction as the x axis. Commands **set xrange**, **set x2range**, **set auto x**, etc will affect both the x and x2 axes.

```
set link x2 via x**2 inverse sqrt(x)
plot "sqrt_data" using 1:2 axes x2y1, "linear_data" using 1:2 axes x1y1
```

This command establishes forward and reverse mapping between the x and x2 axes. The forward mapping is used to generate x2 tic labels and x2 mouse coordinate The reverse mapping is used to plot coordinates given in the x2 coordinate system. Note that the mapping as given is valid only for x non-negative. When mapping to the y2 axis, both <expression1> and <expression2> must use y as dummy variable.

Lmargin

The command **set lmargin** sets the size of the left margin. Please see **set margin** **(p. 167)** for details.

Loadpath

The **loadpath** setting defines additional locations for data and command files searched by the **call**, **load**, **plot** and **splot** commands. If a file cannot be found in the current directory, the directories in **loadpath** 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 platforms. The **show loadpath**, **save** and **save set** commands replace the platform-specific separator with a space character (' ').

If the environment variable GNUPLOT_LIB is set, its contents are appended to **loadpath**. However, **show loadpath** prints the contents of **set loadpath** and GNUPLOT_LIB separately. Also, the **save** and **save set** commands ignore the contents of GNUPLOT_LIB.

Locale

The **locale** 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 command **set locale** `""` will try to determine the locale from the LC_TIME, LC_ALL, or LANG environment variables.

To change the decimal point locale, see **set decimalsign (p. 143)**. To change the character encoding to the current locale, see **set encoding (p. 146)**.

Logscale

Syntax:

```
set logscale <axes> {<base>}
unset logscale <axes>
show logscale
```

where <axes> may be any combinations of **x**, **x2**, **y**, **y2**, **z**, **cb**, and **r** in any order. <base> is the base of the log scaling (default is base 10). If no axes are specified, the command affects all axes except **r**. The command **unset logscale** turns off log scaling for all axes. Note that the ticmarks generated for logscaled axes are not uniformly spaced. See **set xtics (p. 217)**.

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
```

Macros

In this version of gnuplot macro substitution is always enabled. Tokens in the command line of the form @<stringvariablename> will be replaced by the text string contained in <stringvariablename>. See **substitution (p. 57)**.

Mapping

If data are provided to **splot** in spherical or cylindrical coordinates, the **set mapping** 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 **using** entries). The first two are interpreted as the azimuthal and polar angles theta and phi (or "longitude" and "latitude"), in the units specified by **set angles**. 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 **set angles**) 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 **mapping** can be duplicated with the **using** filter on the **splot** command, but **mapping** may be more convenient if many data files are to be processed. However even if **mapping** is used, **using** may still be necessary if the data in the file are not in the required order.

mapping has no effect on **plot**. `world.dem:  mapping demos.`

Margin

The **margin** is the distance between the plot border and the outer edge of the canvas. The size of the margin is chosen automatically, but can be overridden by the **set margin** commands. **show margin** shows the current settings. To alter the distance between the inside of the plot border and the data in the plot itself, see **set offsets (p. 178)**.

Syntax:

```
set lmargin {{at screen} <margin>}
set rmargin {{at screen} <margin>}
set tmargin {{at screen} <margin>}
set bmargin {{at screen} <margin>}
set margins <left>, <right>, <bottom>, <top>
show margin
```

The default 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 can be set using character units.

The keywords **at screen** indicates that the margin is specified as a fraction of the full drawing area. This can be used to precisely line up the corners of individual 2D and 3D graphs in a multiplot. This placement ignores the current values of **set origin** and **set size**, and is intended as an alternative method for positioning graphs within a multiplot.

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.

Micro

By default the "%c" format specifier for scientific notation used to generate axis tick labels uses a lower case u as a prefix to indicate "micro" (10^-6). The **set micro** command tells gnuplot to use a different typographic character (unicode U+00B5). The byte sequence used to represent this character depends on the current encoding. See **format specifiers (p. 149)**, **encoding (p. 146)**.

This command is EXPERIMENTAL. Implementation details may change.

Minussign

Gnuplot uses the C language library routine sprintf() for most formatted input. However it also has its own formatting routine **gprintf()** that is used to generate axis tic labels. The C library routine always use a hyphen character (ascii \055) to indicate a negative number, as in -7. Many people prefer a different typographic minus sign character (unicode U+2212) for this purpose, as in −7. The command

```
set minussign
```

causes gprintf() to use this minus sign character rather than a hyphen in numeric output. In a utf-8 locale this is the multibyte sequence corresponding to unicode U+2212. In a Window codepage 1252 locale this is the 8-bit character ALT+150 ("en dash"). The **set minussign** command will affect axis tic labels and any labels that are created by explicitly invoking gprintf. It has no effect on other strings that contain a hyphen. See **gprintf (p. 149)**.

Note that this command is ignored when you are using any of the LaTeX terminals, as LaTeX has its own mechanism for handling minus signs. It also is not necessary when using the postscript terminal because the postscript prologue output by gnuplot remaps the ascii hyphen code \055 to a different glyph named **minus**.

This command is EXPERIMENTAL. Implementation details may change.

Example (assumes utf8 locale):

```
set minus
A = -5
print "A = ",A                 # printed string will contain a hyphen
print gprintf("A = %g",A)      # printed string will contain character U+2212
set label "V = -5"             # label will contain a hyphen
set label sprintf("V = %g",-5) # label will contain a hyphen
set label gprintf("V = %g",-5) # label will contain character U+2212
```

Monochrome

Syntax:

```
set monochrome {linetype N <linetype properties>}
```

The **set monochrome** command selects an alternative set of linetypes that differ by dot/dash pattern or line width rather than by color. This command replaces the monochrome option offered by certain terminal types in earlier versions of gnuplot. For backward compatibility these terminal types now implicitly invoke "set monochrome" if their own "mono" option is present. For example,

```
set terminal pdf mono
```

is equivalent to

```
set terminal pdf
set mono
```

Selecting monochrome mode does not prevent you from explicitly drawing lines using RGB or palette colors, but see also **set palette gray (p. 184)**. Six monochrome linetypes are defined by default. You can change their properties or add additional monochrome linetypes by using the full form of the command. Changes made to the monochrome linetypes do not affect the color linetypes and vice versa. To restore the usual set of color linetypes, use either **unset monochrome** or **set color**.

Mouse

The command **set mouse** enables mouse actions for the current interactive terminal. It is usually enabled by default in interactive mode, but disabled by default if commands are being read from a file.

There are two mouse modes. The 2D mode works for **plot** commands and for **splot** maps (i.e. **set view** with z-rotation 0, 90, 180, 270 or 360 degrees, including **set view map**). In this mode the mouse position is tracked and you can pan or zoom using the mouse buttons or arrow keys. Some terminals support toggling individual plots on/off by clicking on the corresponding key title or on a separate widget.

For 3D graphs **splot**, the view and scaling of the graph can be changed with mouse buttons 1 and 2, respectively. A vertical motion of Button 2 with the shift key held down changes the **xyplane**. If additionally to these buttons the modifier <ctrl> is held down, the coordinate axes are displayed but the data are suppressed. This is useful for large data sets. Mouse button 3 controls the azimuth of the z axis (see **set view azimuth (p. 211)**).

Mousing is not available inside multiplot mode. When multiplot is completed using **unset multiplot**, then the mouse will be turned on again but acts only on the most recent plot within the multiplot (like replot does).

Syntax:

```
set mouse {doubleclick <ms>} {nodoubleclick}
          {{no}zoomcoordinates}
          {zoomfactors <xmultiplier>, <ymultiplier>}
          {noruler | ruler {at x,y}}
          {polardistance{deg|tan} | nopolardistance}
          {format <string>}
          {mouseformat <int> | <string> | function <f(x,y)>}
          {{no}labels {"labeloptions"}}
          {{no}zoomjump} {{no}verbose}
unset mouse
```

The options **noruler** and **ruler** switch the ruler off and on, the latter optionally setting the origin at the given coordinates. While the ruler is on, the distance in user units from the ruler origin to the mouse is displayed continuously. By default, toggling the ruler has the key binding 'r'.

The option **polardistance** determines if the distance between the mouse cursor and the ruler is also shown in polar coordinates (distance and angle in degrees or tangent (slope)). This corresponds to the default key binding '5'.

Choose the option **labels** to define persistent gnuplot labels using Button 2. The default is **nolabels**, which makes Button 2 draw only a temporary label at the mouse position. Labels are drawn with the current setting of **mouseformat**. The **labeloptions** string is passed to the **set label** command. The default is "point pointstyle 1" which will plot a small plus at the label position. Temporary labels will disappear at the next **replot** or mouse zoom operation. Persistent labels can be removed by holding the Ctrl-Key down while clicking Button 2 on the label's point. The threshold for how close you must be to the label is also determined by the **pointsize**.

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 **bind** command, see help for **bind (p. 52)**.

Note, that user defined **hotkeys** may override the default bindings. See also help for **bind (p. 52)** and **label (p. 161)**.

Doubleclick

The doubleclick resolution is given in milliseconds and used for Button 1, which copies the current mouse position to the **clipboard** on some terminals. The default value is 300 ms. Setting the value to 0 ms triggers the copy on a single click.

Format

The **set mouse format** command specifies a format string for sprintf() which determines how the mouse cursor [x,y] coordinates are printed to the plot window and to the clipboard. The default is "% #g".

This setting is superseded by "set mouse mouseformat".

Mouseformat

Syntax:
```
set mouse mouseformat i
set mouse mouseformat "custom format"
set mouse mouseformat function string_valued_function(x, y)
```

This command controls the format used to report the current mouse position. An integer argument selects one of the format options in the table below. A string argument is used as a format for sprintf() in option 7 and should contain two float specifiers, one for x and one for y.

Use of a custom function returning a string is EXPERIMENTAL. It allows readout of coordinate systems in which inverse mapping from screen coordinates to plot coordinates requires joint consideration of both x and y. See for example the map_projection demo.

Example:
```
'set mouse mouseformat "mouse x,y = %5.2g, %10.3f"'.
```

Use **set mouse mouseformat** "" to turn this string off again.

The following formats are available:
```
0   default (same as 1)
1   axis coordinates                    1.23, 2.45
2   graph coordinates (from 0 to 1)    /0.00, 1.00/
3   x = timefmt     y = axis           [(as set by 'set timefmt'), 2.45]
4   x = date        y = axis           [31. 12. 1999, 2.45]
5   x = time        y = axis           [23:59, 2.45]
6   x = date time   y = axis           [31. 12. 1999 23:59, 2.45]
7   format from 'set mouse mouseformat <format-string>'
8   format from 'set mouse mouseformat function <func>'
```

Scrolling

X and Y axis scaling in both 2D and 3D graphs can be adjusted using the mouse wheel. <wheel-up> scrolls up (increases both YMIN and YMAX by ten percent of the Y range, and increases both Y2MIN and Y2MAX likewise), and <wheel down> scrolls down. <shift-wheel-up> scrolls left (decreases both XMIN and XMAX, and both X2MIN and X2MAX), and <shift-wheel-down> scrolls right. <control-wheel-up> zooms in toward the center of the plot, and <control-wheel-down> zooms out. <shift-control-wheel-up> zooms in along the X and X2 axes only, and <shift-control-wheel-down> zooms out along the X and X2 axes only.

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.

Zoom

Zooming is usually accomplished by holding down the left mouse button and dragging the mouse to delineate a zoom region. Some platforms may require using a different mouse button. The original plot can be restored by typing the 'u' hotkey in the plot window. The hotkeys 'p' and 'n' step back and forth through a history of zoom operations.

The option **zoomcoordinates** determines if the coordinates of the zoom box are drawn at the edges while zooming. This is on by default.

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.

Mttics

Minor tic marks around the perimeter of a polar plot are controlled by by **set mttics**. Please see **set mxtics (p. 173)**.

Multiplot

The command **set multiplot** places **gnuplot** in the multiplot mode, in which several plots are placed next to each other on the same page or screen window.

Syntax:

```
set multiplot
    { title <page title> {font <fontspec>} {enhanced|noenhanced} }
    { layout <rows>,<cols>
      {rowsfirst|columnsfirst} {downwards|upwards}
      {scale <xscale>{,<yscale>}} {offset <xoff>{,<yoff>}}
      {margins <left>,<right>,<bottom>,<top>}
      {spacing <xspacing>{,<yspacing>}}
    }
set multiplot {next|previous}
unset multiplot
```

For some terminals, no plot is displayed until the command **unset multiplot** 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.

The **clear** command is used to erase the rectangular area of the page that will be used for the next plot. This is typically needed to inset a small plot inside a larger plot.

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 **replot** commands within the **set multiplot**/**unset multiplot** block.

The multiplot title is separate from the individual plot titles, if any. Space is reserved for it at the top of the page, spanning the full width of the canvas.

The commands **set origin** and **set size** must be used to correctly position each plot if no layout is specified or if fine tuning is desired. See **set origin (p. 178)** and **set size (p. 195)** for details of their usage.

Example:

```
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
```

This displays a plot of cos(x) stacked above a plot of sin(x).

set size and **set origin** refer to the entire plotting area used for each plot. Please also see **set term size (p. 31)**. If you want to have the axes themselves line up, you can guarantee that the margins are the same size with the **set margin** commands. See **set margin (p. 167)** 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 **set size** and **set origin** 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**. The commands **set multiplot next** and **set multiplot previous** are relevant only in the context of using the layout option. **next** skips the next position in the grid, leaving a blank space. **prev** returns to the grid position immediately preceding the most recently plotted position.

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. **unset multiplot** will turn off the automatic layout and restore the values of **set size** and **set origin** 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.

Another possibility is to set uniform margins for all plots in the layout with options **layout margins** and **spacing**, which must be used together. With **margins** you set the outer margins of the whole multiplot grid.

spacing gives the gap size between two adjacent subplots, and can also be given in **character** or **screen** units. If a single value is given, it is used for both x and y direction, otherwise two different values can be selected.

If one value has no unit, the one of the preceding margin setting is used.

Example:

```
set multiplot layout 2,2 margins 0.1, 0.9, 0.1, 0.9 spacing 0.0
```

In this case the two left-most subplots will have left boundaries at screen coordinate 0.1, the two right-most subplots will have right boundaries at screen coordinate 0.9, and so on. Because the spacing between subplots is given as 0, their inner boundaries will superimpose.

Example:

```
set multiplot layout 2,2 margins char 5,1,1,2 spacing screen 0, char 2
```

This produces a layout in which the boundary of both left subplots is 5 character widths from the left edge of the canvas, the right boundary of the right subplots is 1 character width from the canvas edge. The overall bottom margin is one character height and the overall top margin is 2 character heights. There is no horizontal gap between the two columns of subplots. The vertical gap between subplots is equal to 2 character heights.

Example:

```
set multiplot layout 2,2 columnsfirst margins 0.1,0.9,0.1,0.9 spacing 0.1
set ylabel 'ylabel'
plot sin(x)
set xlabel 'xlabel'
plot cos(x)
unset ylabel
unset xlabel
plot sin(2*x)
set xlabel 'xlabel'
plot cos(2*x)
unset multiplot
```

See also multiplot demo (multiplt.dem)

Mx2tics

Minor tic marks along the x2 (top) axis are controlled by **set mx2tics**. Please see **set mxtics** **(p. 173)**.

Mxtics

Minor tic marks along the x axis are controlled by **set mxtics**. They can be turned off with **unset mxtics**. Similar commands control minor tics along the other axes.

Syntax:

```
set mxtics {<freq> | default}
unset mxtics
show mxtics
```

The same syntax applies to **mytics**, **mztics**, **mx2tics**, **my2tics**, **mrtics**, **mttics** 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 ytics format ""
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 **set xtics (p. 217)** for information about these.

My2tics

Minor tic marks along the y2 (right-hand) axis are controlled by **set my2tics**. Please see **set mxtics (p. 173)**.

Mytics

Minor tic marks along the y axis are controlled by **set mytics**. Please see **set mxtics (p. 173)**.

Mztics

Minor tic marks along the z axis are controlled by **set mztics**. Please see **set mxtics (p. 173)**.

Nonlinear

Syntax:

```
set nonlinear <axis> via f(axis) inverse g(axis)
unset nonlinear <axis>
```

This command is similar to the **set link** command except that only one of the two linked axes is visible. The hidden axis remains linear. Coordinates along the visible axis are mapped by applying g(x) to hidden axis coordinates. f(x) maps the visible axis coordinates back onto the hidden linear axis. You must provide both the forward and inverse expressions.

To illustrate how this works, consider the case of a log-scale x2 axis.

```
set x2ange [1:1000]
set nonlinear x2 via log10(x) inverse 10**x
```

This achieves the same effect as **set log x2**. The hidden axis in this case has range [0:3], obtained by calculating [log10(xmin):log10(xmax)].

The transformation functions f() and g() must be defined using a dummy variable appropriate to the nonlinear axis:

```
axis: x x2   dummy variable x
axis: y y2   dummy variable y
axis: z cb   dummy variable z
axis: r      dummy variable r
```

Example:

```
set xrange [-3:3]
set nonlinear x via norm(x) inverse invnorm(x)
```

This example establishes a probability-scaled ("probit") x axis, such that plotting the cumulative normal function Phi(x) produces a straight line plot against a linear y axis.

Example:

```
logit(p) = log(p/(1-p))
logistic(a) = 1. / (1. + exp(-a))
set xrange [.001 : .999]
set nonlinear y via logit(y) inverse logistic(y)
plot logit(x)
```

This example establishes a logit-scaled y axis such that plotting logit(x) on a linear x axis produces a straight line plot.

Example:

```
f(x) = (x <= 100) ? x : (x < 500) ? NaN : x-390
g(x) = (x <= 100) ? x : x+390
set xrange [0:1000] noextend
set nonlinear x via f(x) inverse g(x)
set xtics add (100,500)
plot sample [x=1:100] x, [x=500:1000] x
```

This example creates a "broken axis". X coordinates 0-100 are at the left, X coordinates 500-1000 are at the right, there is a small gap (10 units) between them. So long as no data points with (100 < x < 500) are plotted, this works as expected.

Object

The **set object** command defines a single object which will appear in subsequent plots. You may define as many objects as you like. Currently the supported object types are **rectangle**, **circle**, **ellipse**, and **polygon**. Rectangles inherit a default set of style properties (fill, color, border) from those set by the command **set style rectangle**, but each object can also be given individual style properties. Circles, ellipses, and polygons inherit the fill style from **set style fill**. Objects to be drawn in 2D plots may be defined in any combination of axis, graph, polar, or screen coordinates.

Object specifications in 3D plots cannot use graph coordinates. Rectangles and ellipses in 3D plots are limited to screen coordinates.

Syntax:

```
set object <index>
    <object-type> <object-properties>
    {front|back|behind|depthorder}
    {clip|noclip}
    {fc|fillcolor <colorspec>} {fs <fillstyle>}
    {default} {lw|linewidth <width>} {dt|dashtype <dashtype>}
unset object <index>
```

<object-type> is either **rectangle**, **ellipse**, **circle**, or **polygon**. Each object type has its own set of characteristic properties.

The options **front**, **back**, **behind** control whether the object is drawn before or after the plot itself. See **layers (p. 51)**. Setting **front** will draw the object in front of all plot elements, but behind any labels that are also marked **front**. Setting **back** will place the object behind all plot curves and labels. Setting **behind** will place the object behind everything including the axes and **back** rectangles, thus

```
set object rectangle from screen 0,0 to screen 1,1 behind
```

can be used to provide a colored background for the entire graph or page.

By default, objects are clipped to the graph boundary unless one or more vertices are given in screen coordinates. Setting **noclip** will disable clipping to the graph boundary, but will still clip against the screen size.

The fill color of the object is taken from the <colorspec>. **fillcolor** may be abbreviated **fc**. The fill style is taken from <fillstyle>. See **colorspec (p. 49)** and **fillstyle (p. 199)**. If the keyword **default** is given, these properties are inherited from the default settings at the time a plot is drawn. See **set style rectangle (p. 202)**.

Rectangle

Syntax:

```
set object <index> rectangle
    {from <position> {to|rto} <position> |
     center <position> size <w>,<h> |
     at <position> size <w>,<h>}
```

The position of the rectangle may be specified by giving the position of two diagonal corners (bottom left and top right) or by giving the position of the center followed by the width and the height. In either case the positions may be given in axis, graph, or screen coordinates. See **coordinates (p. 32)**. The options **at** and **center** are synonyms.

Examples:

```
# Force the entire area enclosed by the axes to have background color cyan
set object 1 rect from graph 0, graph 0 to graph 1, graph 1 back
set object 1 rect fc rgb "cyan" fillstyle solid 1.0

# Position a red square with lower left at 0,0 and upper right at 2,3
set object 2 rect from 0,0 to 2,3 fc lt 1

# Position an empty rectangle (no fill) with a blue border
set object 3 rect from 0,0 to 2,3 fs empty border rgb "blue"

# Return fill and color to the default style but leave vertices unchanged
set object 2 rect default
```

Rectangle corners specified in screen coordinates may extend beyond the edge of the current graph. Otherwise the rectangle is clipped to fit in the graph.

Ellipse

Syntax:

```
set object <index> ellipse {at|center} <position> size <w>,<h>
    {angle <orientation>} {units xy|xx|yy}
    {<other-object-properties>}
```

The position of the ellipse is specified by giving the center followed by the width and the height (actually the major and minor axes). The keywords **at** and **center** are synonyms. The center position may be given in axis, graph, or screen coordinates. See **coordinates (p. 32)**. The major and minor axis lengths must be given in axis coordinates. The orientation of the ellipse is specified by the angle between the horizontal axis and the major diameter of the ellipse. If no angle is given, the default ellipse orientation will be used instead (see **set style ellipse (p. 202)**). The **units** keyword controls the scaling of the axes of the ellipse.

units xy means that the major axis is interpreted in terms of units along the x axis, while the minor axis in that of the y axis. **units xx** means that both axes of the ellipses are scaled in the units of the x axis, while **units yy** means that both axes are in units of the y axis. The default is **xy** or whatever **set style ellipse units** was set to.

NB: If the x and y axis scales are not equal, (e.g. **units xy** is in effect) then the major/minor axis ratio will no longer be correct after rotation.

Note that **set object ellipse size <2r>,<2r>** does not in general produce the same result as **set object circle <r>**. The circle radius is always interpreted in terms of units along the x axis, and will always produce a circle even if the x and y axis scales are different and even if the aspect ratio of your plot is not 1. If **units** is set to **xy**, then 'set object ellipse' interprets the first <2r> in terms of x axis units and the second <2r> in terms of y axis units. This will only produce a circle if the x and y axis scales are identical and the plot aspect ratio is 1. On the other hand, if **units** is set to **xx** or **yy**, then the diameters specified in the 'set object' command will be interpreted in the same units, so the ellipse will have the correct aspect ratio, and it will maintain its aspect ratio even if the plot is resized.

Circle

Syntax:

```
set object <index> circle {at|center} <position> size <radius>
    {arc [<begin>:<end>]} {no{wedge}}
    {<other-object-properties>}
```

The position of the circle is specified by giving the position of the center center followed by the radius. The keywords **at** and **center** are synonyms. In 2D plots the position and radius may be given in any coordinate system. See **coordinates (p. 32)**. Circles in 3D plots cannot use graph coordinates. In all cases the radius is calculated relative to the horizontal scale of the axis, graph, or canvas. Any disparity between the horizontal and vertical scaling will be corrected for so that the result is always a circle. If you want to draw a circle in plot coordinates (such that it will appear as an ellipse if the horizontal and vertical scales are different), use **set object ellipse** instead.

By default a full circle is drawn. The optional qualifier **arc** specifies a starting angle and ending angle, in degrees, for one arc of the circle. The arc is always drawn counterclockwise.

See also **set style circle (p. 202)**, **set object ellipse (p. 176)**.

Polygon

Syntax:

```
set object <index> polygon
    from <position> to <position> ... {to <position>}
```

or

```
    from <position> rto <position> ... {rto <position>}
```

The position of the polygon may be specified by giving the position of a sequence of vertices. These may be given in any coordinate system. If relative coordinates are used (rto) then the coordinate type must match that of the previous vertex. See **coordinates (p. 32)**.

Example:

```
set object 1 polygon from 0,0 to 1,1 to 2,0
set object 1 fc rgb "cyan" fillstyle solid 1.0 border lt -1
```

Depthorder The option **set object N depthorder** applies to 3D polygon objects only. Rather than assigning the object to layer front/back/behind it is included in the list of pm3d quadrangles sorted and rendered in order of depth by **set pm3d depthorder**. As with pm3d surfaces, two-sided coloring can be generated by specifying the object fillcolor as a linestyle. In this case the ordering of the first three vertices in the polygon determines the "side".

If you set this property for an object that is not a 3D polygon it probably will not be drawn at all.

Offsets

Autoscaling sets the x and y axis ranges to match the coordinates of the data that is plotted. Offsets provide a mechanism to expand these ranges to leave empty space between the data and the plot borders. Autoscaling then further extends each range to reach the next axis tic unless this has been suppressed by **set autoscale noextend** or **set xrange noextend**. See **noextend (p. 132)**. Offsets affect only scaling for the x1 and y1 axes.

Syntax:

```
set offsets <left>, <right>, <top>, <bottom>
unset offsets
show offsets
```

Each offset may be a constant or an expression. Each defaults to 0. By default, the left and right offsets are given in units of the first x axis, the top and bottom offsets in units of the first y axis. Alternatively, you may specify the offsets as a fraction of the total graph dimension by using the keyword "graph". Only "graph" offsets are possible for nonlinear axes.

A positive offset expands the axis range in the specified direction, e.g. a positive bottom offset makes ymin more negative. Negative offsets interact badly with autoscaling and clipping.

Example:

```
set autoscale noextend
set offsets graph 0.05, 0, 2, 2
plot sin(x)
```

This graph of sin(x) will have y range [-3:3] because the function will be autoscaled to [-1:1] and the vertical offsets add 2 at each end of the range. The x range will be [-11:10] because the default is [-10:10] and it has been expanded to the left by 0.05 of that total range.

Origin

The **set origin** 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 (p. 32)** for information about this system).

Syntax:

```
set origin <x-origin>,<y-origin>
```

Output

By default, screens are displayed to the standard output. The **set output** 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 **set output** 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** or **wxt**, ignore **set output**.])

When both **set terminal** and **set output** 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 a separate open command for binary files.

On platforms that support pipes, it may be useful to pipe terminal output. For instance,

```
set output "|lpr -Plaser filename"
set term png; set output "|display png:-"
```

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.

Overflow

Syntax:

```
set overflow {float | NaN | undefined}
unset overflow
```

This version of gnuplot supports 64-bit integer arithmetic. This means that for values from 2^53 to 2^63 (roughly 10^16 to 10^19) integer evaluation preserves more precision than evaluation using IEEE 754 floating point arithmetic. However unlike the IEEE floating point representation, which sacrifices precision to span a total range of roughly [-10^307 : 10^307], integer operations that result in values outside the range [-2^63 : 2^63] overflow. The **set overflow** command lets you control what happens in case of overflow. See options below.

set overflow is the same as **set overflow float**. It causes the result to be returned as a real number rather than as an integer. This is the default.

The command **unset overflow** causes integer arithmetic overflow to be ignored. No error is shown. This may be desirable if your platform allows only 32-bit integer arithmetic and you want to approximate the behaviour of gnuplot versions prior to 5.4.

The **reset** command does not affect the state of overflow handling.

Earlier gnuplot versions were limited to 32-bit arithmetic and ignored integer overflow. Note, however, that some built-in operators did not use integer arithmetic at all, even when given integer arguments. This included the exponentiation operator N**M and the summation operator (see **summation (p. 42)**). These operations now return an integer value when given integer arguments, making them potentially susceptible to overflow and thus affected by the state of **set overflow**.

Float

If an integer arithmetic expression overflows the limiting range, [-2^63 : 2^63] for 64-bit integers, the result is returned as a floating point value instead. This is not treated as an error. Example:

```
gnuplot> set overflow float
gnuplot> A = 2**62 - 1;  print A, A+A, A+A+A
4611686018427387903 9223372036854775806 1.38350580552822e+19
```

NaN

If an integer arithmetic expression overflows the limiting range, [-2^63 : 2^63] for 64-bit integers, the result is returned as NaN (Not a Number). This is not treated as an error. Example:

```
gnuplot> set overflow NaN
gnuplot> print 10**18, 10**19
1000000000000000000 NaN
```

Undefined

If an integer arithmetic expression overflows the limiting range, [-2^63 : 2^63] for 64-bit integers, the result is undefined. This is treated as an error. Example:

```
gnuplot> set overflow undefined
gnuplot> A = 10**19
                ^
         undefined value
```

Affected operations

The **set overflow** state affects the arithmetic operators

```
+ - * / **
```

and the built-in summation operation **sum**.

All of these operations will return an integer result if all of the arguments are integers, so long as no overflow occurs during evaluation.

The **set overflow** state does not affect logical or bit operations

```
<< >>  | ^ &
```

If overflow occurs at any point during the course of evaluating of a summation **set overflow float** will cause the result to be returned as a real number even if the final sum is within the range of integer representation.

Palette

The palette is a set of colors, usually ordered in the form of one or more stepped gradients, used for **pm3d** surfaces and other graph elements colored by z value. Colors in the current palette are automatically mapped from plot coordinates z values or an extra data column of gray values. Palette colors also can be accessed explicitly in a color specification (see **colorspec (p. 49)**)

- as a **gray value** also known as **palette fraction** in the range [0:1]
- as a **z value** corresponding to the z coordinate of a plot element
- as a **cb value** in the range [cbmin:cbmax] (see **set cbrange (p. 225)**)

The current palette is shown by default in a separate **colorbox** drawn next to plots that use plot style **pm3d**. The colorbox can be manually selected or disabled. See **set colorbox (p. 139)**.

Syntax:

```
 set palette
 set palette {
           { gray | color }
           { gamma <gamma> }
           {   rgbformulae <r>,<g>,<b>
             | defined { ( <gray1> <color1> {, <grayN> <colorN>}... ) }
             | file '<filename>' {datafile-modifiers}
             | functions <R>,<G>,<B>
           }
```

```
                { cubehelix {start <val>} {cycles <val>} {saturation <val>} }
                { model { RGB | HSV | CMY } }
                { 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 colornames
```

set palette (i.e. without options) sets up the default values. Otherwise, the options can be given in any order. **show palette** shows the current palette properties.

show palette gradient displays the gradient defining the palette (if appropriate). **show palette rgbformulae** prints the available fixed gray –> color transformation formulae. **show colornames** prints the known color names.

show palette palette <n> prints to the screen or to the file given by **set print** a 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. Alternatively, the **test palette** command will plot the R,G,B profiles for the current palette and leave the profile values in a datablock $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. **palette rgbformulae** and **palette functions** use analytic formulae whereas **palette defined** and **palette file** use interpolated tables. **palette rgbformulae** reduces the size of postscript output to a minimum.

The command **show palette fit2rgbformulae** finds the best matching **set palette rgbformulae** for the current **set palette**. 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. **set palette rgbformulae**, **set palette defined**, **set palette file** 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 (**set palette gray**) and for the **cubehelix** color palette schemes. Gamma = 1 produces a linear ramp of intensity. See **test palette (p. 233)**.

Many terminals support only discrete number of colors (e.g. 256 colors in gif). After the default gnuplot linetype colors are allocated, the rest of the available colors are by default reserved for pm3d. Thus a multiplot using multiple palettes could fail because the first palette has used all the available color positions. You can mitigate this limitation by using **set palette maxcolors <N>** with a reasonably small value of N. This option causes N discrete colors to be selected from a continuous palette sampled at equally spaced intervals. If you want unequal spacing of N discrete colors, use **set palette defined** instead of a single continuous palette.

RGB color space might not be the most useful color space to work in. For that reason you may change the color space **model** to one of **RGB**, **HSV**, **CMY**. 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**, or **C**, 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, Blue; CMY stands for Cyan, Magenta, Yellow; HSV stands for Hue, Saturation, Value. For more information on color models see: http://en.wikipedia.org/wiki/Color_space

Note: Earlier gnuplot versions accepted YIQ and XYZ color space models also, but the implementation was never complete or correct.

Rgbformulae

For **rgbformulae** three suitable mapping functions have to be chosen. This is done via **rgbformulae <r>,<g>,<b>**. The available mapping functions are listed by **show palette rgbformulae**. 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** **(p. 182)** and **negative** **(p. 182)** 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 **rgbformulae** 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
```

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 must specify the gray values and the corresponding RGB values between which linear interpolation will be done.

Syntax:

```
set palette  defined { ( <gray1> <color1> {, <grayN> <colorN>}... ) }
```

<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 **show colornames** 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 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 "yellow", 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 defined ( 0 0 1 0, 1 0 1 1, 6 0.8333 1 1, 7 0.8333 0 1)
```

Approximate the default palette used by MATLAB:

```
set pal defined (1 '#00008f', 8 '#0000ff', 24 '#00ffff', \
                 40 '#ffff00', 56 '#ff0000', 64 '#800000')
```

To produce a palette with only a few, equally-spaced colors:

```
set palette model RGB maxcolors 4
set palette defined ( 0 "yellow", 1 "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" )
```

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 RGB functions 1.1*gray**0.25, gray**0.75, 0
```

A gamma-corrected black and white palette

```
gamma = 2.2
color(gray) = gray**(1./gamma)
set palette model RGB functions color(gray), color(gray), color(gray)
```

Gray

set palette gray switches to a grayscale palette shading from 0.0 = black to 1.0 = white. **set palette color** is an easy way to switch back from the gray palette to the last color mapping.

Cubehelix

The "cubehelix" option defines a family of palettes in which color (hue) varies along the standard color wheel while at the same time the net intensity increases monotonically as the gray value goes from 0 to 1.

```
D A Green (2011) http://arxiv.org/abs/1108.5083
```

start defines the starting point along the color wheel in radians. **cycles** defines how many color wheel cycles span the palette range. Larger values of **saturation** produce more saturated color; saturation > 1 may lead to clipping of the individual RGB components and to intensity becoming non-monotonic. The palette is also affected by **set palette gamma**. The default values are

```
set palette cubehelix start 0.5 cycles -1.5 saturation 1
set palette gamma 1.5
```

File

set palette file 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 **using** 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
```

Binary palette files are supported as well, see **binary general** **(p. 103)**. Example: put 64 triplets of R,G,B doubles into file palette.bin and load it by

```
set palette file "palette.bin" binary record=64 using 1:2:3
```

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.

The gamma correction is applied to the cubehelix color palette family, but not to other palette coloring schemes. However, you may easily implement gamma correction for explicit color functions.

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.

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 **multiplot** with fixed **origin** and **size**.

If you are writing a pm3d surface to a postscript file, it may be possible to reduce the file size by up to 50% by the enclosed awk script **pm3dCompress.awk**. If the data lies on a rectangular grid, even greater compression may be possible using the script **pm3dConvertToImage.awk**. Usage:

```
awk -f pm3dCompress.awk thefile.ps >smallerfile.ps
awk -f pm3dConvertToImage.awk thefile.ps >smallerfile.ps
```

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 2D plotting, a parametric function is determined by a pair of parametric functions operating on a parameter. An example of a 2D parametric function would be **plot sin(t),cos(t)**, which draws a circle (if the aspect ratio is set correctly — see **set size (p. 195)**). **gnuplot** will display an error message if both functions are not provided for a parametric **plot**.

For 3D 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 3D 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, 3D 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 **set trange**, **set urange**, and **set vrange**, 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.

Paxis

Syntax:

```
set paxis <axisno> {range <range-options> | tics <tic-options>}
set paxis <axisno> label <label-options> { offset <radial-offset> }
show paxis <axisno> {range | tics}
```

The **set paxis** command is equivalent to the **set xrange** and **set xtics** commands except that it acts on one of the axes p1, p2, ... used in parallel axis plots and spiderplots. See **parallelaxes (p. 76)**, **set xrange (p. 215)**, and **set xtics (p. 217)**. The normal options to the range and tics commands are accepted although not all options make sense for parallel axis plots.

set paxis <axisno> label <label-options> is relevant to spiderplots but ignored otherwise. Axes of a parallel axis plot can be labeled using the **title** option of the plot command, which generates an xtic label. Note that this may require also **set xtics**.

The axis linetype properties are controlled using **set style parallelaxis (p. 203)**.

Pixmap

Syntax:

```
set pixmap <index> "filename" at <position>
           {width <w> | height <h> | size <w>,<h>}
           {front|back|behind} {center}
show pixmaps
unset pixmaps
unset pixmap <index>
```

The **set pixmap** command is similar to **set object** in that it defines an object that will appear on subsequent plots. The rectangular array of red/green/blue/alpha values making up the pixmap are read from a png, jpeg, or gif file. The position and extent occupied by the pixmap in the gnuplot output may be specified in any coordinate system (see **coordinates (p. 32)**). The coordinates given by **at <position>** refer to the lower left corner of the pixmap unless keyword **center** is present.

If the x-extent of the rendered pixmap is set using **width <x-extent>** the aspect ratio of the original image is retained and neither the aspect ratio nor the orientation of the pixmap changes with axis scaling or rotation. Similarly if the y-extent is set using **height <y-extent>**. If both the x-extent and y-extent are given using **size** <**x-extent**> <**y-extent**> this overrides the original aspect ratio. If no size is set then the original size in pixels is used (the effective size is then terminal-dependent).

Pixmaps are not clipped to the border of the plot. As an exception to the general behaviour of objects and layers, a pixmap assigned to layer **behind** is rendered for only the first plot in a multiplot. This allows all panels in a multiplot to share a single background pixmap.

Examples:

```
# Use a gradient as the background for all plotting
# Both x and y will be resized to fill the entire canvas
set pixmap 1 "gradient.png"
set pixmap 1 at screen 0, 0 size screen 1, 1 behind

# Place a logo at the lower right of each page plotted
set pixmap 2 "logo.jpg"
set pixmap 2 at screen 0.95, 0 width screen 0.05 behind

# Place a small image at some 3D coordinate
# It will move as if attached to the surface being plotted
# but will always face forward and remain upright
set pixmap 3 "image.png" at my_x, my_y, f(my_x,my_y) width screen .05
splot f(x,y)
```

Plot

The **show plot** command shows the current plotting command as it results from the last **plot** and/or **splot** and possible subsequent **replot** commands.

In addition, the **show plot add2history** command adds this current plot command into the **history**. It is useful if you have used **replot** to add more curves to the current plot and you want to edit the whole command now.

Pm3d

pm3d is an **splot** style for drawing palette-mapped 3d and 4d data as color/gray maps and surfaces. It allows plotting gridded or non-gridded data without preprocessing. pm3d style options also affect solid-fill polygons used to construct other 3D plot elements.

Syntax (the options can be given in any order):

```
set pm3d {
           { at <position> }
           { interpolate <steps/points in scan, between scans> }
           { scansautomatic | scansforward | scansbackward
                            | depthorder {base} }
           { flush { begin | center | end } }
           { ftriangles | noftriangles }
           { clip {z} | clip1in | clip4in }
           { {no}clipcb }
           { corners2color
             { mean|geomean|harmean|rms|median|min|max|c1|c2|c3|c4 }
           }
           { {no}lighting
             {primary <fraction>} {specular <fraction>} {spec2 <fraction>}
           }
           { border {<linestyle-options>}}
```

```
              { implicit | explicit }
              { map }
            }
     show pm3d
     unset pm3d
```

Note that pm3d plots are plotted sequentially in the order given in the splot command. Thus earlier plots may be obscured by later plots. To avoid this you can use the **depthorder** scan option.

The pm3d surfaces can be projected onto the **top** or **bottom** of the view box. See **pm3d position (p. 189)**. The following command draws 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 **set palette (p. 180)**, **set cbrange (p. 225)**, **set colorbox (p. 139)**, and the demo file **demo/pm3d.dem**.

Implicit

A pm3d color surface is drawn if the splot command specifies **with pm3d**, if the data or function **style** is set to pm3d globally, or if the pm3d mode is **set pm3d implicit**. In the latter two cases, the pm3d surface is draw in addition to the mesh produced by the style specified in the plot command. E.g.

```
splot 'fred.dat' with lines, 'lola.dat' with lines
```

would draw both a mesh of lines and a pm3d surface for each data set. If the option **explicit** is on (or **implicit** is off) only plots specified by the **with 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 (only) and 'lola.dat' with a pm3d surface.

On gnuplot start-up, the mode is **explicit**. For historical and compatibility reasons, the commands **set pm3d;** (i.e. no options) and **set pm3d at X ...** (i.e. **at** is the first option) change the mode to **implicit**. The command **set pm3d;** sets other options to their default state.

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.

Algorithm

Let us first describe how a map/surface is drawn. The input data come from an evaluated function or from an **splot data file**. 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 **using (p. 115)**) 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 **zrange** or **cbrange** 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 **set palette (p. 180)** 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 (p. 139)**, **set cbrange (p. 225)**.

Lighting

By default the colors assigned to pm3d objects are not dependent on orientation or viewing angle. This state corresponds to **set pm3d nolighting**. The command **set pm3d lighting** selects a simple lighting model consisting of a single fixed source of illumination contributing 50% of the overall lighting. The strength of this light relative to the ambient illumination can be adjusted by **set pm3d lighting primary <fraction>**. Inclusion of specular highlighting can be adjusted by setting a fractional contribution:

```
set pm3d lighting primary 0.50 specular 0.0   # no highlights
set pm3d lighting primary 0.50 specular 0.6   # strong highlights
```

Solid-color pm3d surfaces tend to look very flat without specular highlights. Since the highlights from a single source only affect one side of the surface, a second spotlight source may be desirable to add specular highlights from the opposite direction. This is controlled by "spec2 <contribution>". EXPERIMENTAL (details may change in a future version): The second spotlight is a pure red light source that by default contributes nothing (spec2 0.0). See also hidden_compare.dem `(comparison of hidden3d and pm3d treatment of solid-color surfaces)`

Position

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. On the other hand, the **depthorder** option completely reorders the quadrangles. The rendering is performed after a depth sorting, which allows to visualize even complicated surfaces; see **pm3d depthorder (p. 189)** for more details.

Scanorder

```
set pm3d {scansautomatic | scansforward | scansbackward | depthorder}
```

By default the quadrangles making up a pm3d solid surface are rendered in the order they are encountered along the surface grid points. This order may be controlled by the options

scansautomatic|**scansforward**|**scansbackward**. These scan options are not in general compatible with hidden-surface removal.

If two successive 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 scans. This can be used to draw a smooth map boundary.

Gnuplot does not do true hidden surface removal for solid surfaces, but often it is sufficient to render the component quadrangles in order from furthest to closest. This mode may be selected using the option

```
set pm3d depthorder
```

Note that the global option **set hidden3d** does not affect pm3d surfaces.

The **depthorder** option by itself tends to produce bad results when applied to the long thin rectangles generated by **splot with boxes**. It works better to add the keyword **base**, which performs the depth sort using the intersection of the box with the plane at z=0. This type of plot is further improved by adding a lighing model. Example:

```
set pm3d depthorder base
set pm3d lighting
set boxdepth 0.4
splot $DATA using 1:2:3 with boxes
```

Clipping

Syntax:

```
set pm3d {clip {z} | clip1in | clip4in}
set pm3d {no}clipcb
```

The component quadrangles of a pm3d surface or other 3D object are by default smoothly clipped against the current zrange. This is a change from earlier gnuplot versions.

Alternatively, surfaces can be clipped by rendering whole quadrangles but only those with all 4 corners in-range on x, y, and z (**set pm3d clip4in**), or only those with at least one corner in-range on x, y, and z (**set pm3d clip1in**). The options **clip**, **clip1in**, and **clip4in** are mutually exclusive.

Separate from clipping based on spatial x, y, and z coordinates, quadrangles can be rendered or not based on extreme palette color values. **clipcb**: (default) palette color values < cbmin are clipped to equal cbmin; palette color values > cbmax are clipped to equal cbmax. **noclipcb**: quadrangles with color value outside cbrange are not drawn at all.

Color_assignment

The default pm3d coloring assigns an individual color to each quadrangle of the surface grid. For alternative coloring schemes that assign uniform color to the entire surface, see **pm3d fillcolor (p. 191)**.

A single gray/color value (i.e. not a gradient) is assigned to each quadrangle. This value is calculated from the z-coordinates the four quadrangle corners according to **corners2color <option>**. The value is then used to select a color from the current palette. See **set palette (p. 180)**. It is not possible to change palettes inside a single **splot** command.

If a fourth column of data is provided, the coloring of individual quadrangles works as above except that the color value is distinct from the z value. As a separate coloring option, the fourth data column may provide instead an RGB color. See **rgbcolor variable (p. 50)**. In this case the plotting command must be

```
splot ... using 1:2:3:4 with pm3d lc rgb variable
```

Notice that ranges of z-values and color-values for surfaces are adjustable independently by **set zrange**, **set cbrange**, **set log z**, **set log cb**, etc.

Corners2color

The color of each quadrangle in a pm3d surface is assigned based on the color values of its four bounding vertices. The options 'mean' (default), 'geomean', 'harmean, 'rms', 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.

Border

The option **set pm3d border {line-properties}** draws bounding lines around each quadrangle as it is rendered. Normally this is used in conjunction with the **depthorder** option to approximate hidden line removal. Note that the global option **set hidden3d** has no effect on pm3d plots. Default line properties (color, width) optionally follow the keyword **border**. These defaults can be overridden later in an splot command.

Example of recommended usage:

```
set pm3d at s depthorder border lw 0.2 lt black
unset hidden3d
unset surf
splot x*x+y*y linecolor rgb "blue"   # otherwise it would be black
```

Fillcolor

```
splot FOO with pm3d fillcolor <colorspec>
```

Plot style **with pm3d** accepts an optional fillcolor in the splot command. This specification is applied to the entire pm3d surface. See **colorspec (p. 49)**. Most fillcolor specifications will result in a single solid color, which is hard to interpret visually unless there is also a lighting model present to distinguish surface components based on orientation. See **pm3d lighting (p. 189)**.

There are a few special cases. **with pm3d fillcolor palette** would produce the same result as the default pm3d palette-based coloring, and is therefore not a useful option. **with pm3d fillcolor linestyle N** is more interesting. This variant assigns distinct colors to the top and bottom of the pm3d surface, similar to the color scheme used by gnuplot's **hidden3d** mode. Linestyle N is used for the top surface; linestyle N+1 for the bottom surface. Note that "top" and "bottom" depend on the scan order, so that the colors are inverted for **pm3d scansbackward** as compared to **pm3d scansforward**. This coloring option works best with **pm3d depthorder**, however, which unfortunately does not allow you to control the scan order so you may have to instead swap the colors defined for linestyles N and N+1.

Interpolate

The option **interpolate m,n** will interpolate between grid points to generate a finer mesh. For data files, this smooths the color surface and enhances the contrast of spikes in the surface. When working with functions, interpolation makes little sense. It would usually make more sense to increase **samples** and **isosamples**.

For positive m and n, each quadrangle or triangle is interpolated m-times and n-times in the respective direction. For negative m and n, the interpolation frequency is chosen so that there will be at least |m| and |n| points drawn; you can consider this as a special gridding function.

Note: **interpolate 0,0**, will automatically choose an optimal number of interpolated surface points.

Note: Currently color interpolation is always linear, even if corners2color is set to a nonlinear scheme such as the geometric mean.

Deprecated_options

The deprecated option **set pm3d map** was equivalent to **set pm3d at b; set view map; set style data pm3d; set style func pm3d;**

The deprecated option **set pm3d hidden3d N** was equivalent to **set pm3d border ls N**.

Pointintervalbox

The **pointinterval** and **pointnumber** properties of a line type are used only in plot style **linespoints**. A negative value of pointinterval or pointnumber, e.g. -N, means that before the selected set of point symbols are drawn a box (actually circle) behind each point symbol is blanked out by filling with the background color. The command **set pointintervalbox** controls the radius of this blanked-out region. It is a multiplier for the default radius, which is equal to the point size.

Pointsize

The **set pointsize** 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 **plot with (p. 124)** for details.

Please note that the pointsize setting is not supported by all terminal types.

Polar

The **set polar** command changes the meaning of the plot from rectangular coordinates to polar coordinates.

Syntax:

```
set polar
unset polar
show polar
```

In polar coordinates, the dummy variable (t) represents an angle theta. The default range of t is [0:2*pi], or [0:360] if degree units have been selected (see **set angles (p. 129)**).

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 **set mapping (p. 166)** command for similar functionality for **splot (p. 225)**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. The r, x and y ranges control the extent of the graph in the x and y directions. Each of these ranges, as well as the rrange, may be autoscaled or set explicitly. For details, see **set rrange (p. 194)** and **set xrange (p. 215)**.

Example:

```
set polar
plot t*sin(t)
set trange [-2*pi:2*pi]
set rrange [0:3]
plot 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 the area within 3 units of the origin. This has the effect of limiting x and y to [-3:3].

By default polar plots are oriented such that theta=0 is at the far right, with theta increasing counterclockwise. You can change both the origin and the sense explicitly. See **set theta (p. 206)**.

You may want to **set size square** to have **gnuplot** try to make the aspect ratio equal to unity, so that circles look circular. Tic marks around the perimeter can be specified using **set ttics**. See also `polar demos (polar.dem)`

and `polar data plot (poldat.dem).`

Print

The **set print** command redirects the output of the **print** command to a file.

Syntax:

```
set print
set print "-"
set print "<filename>" [append]
set print "|<shell_command>"
set print $datablock [append]
```

set print with no parameters restores output 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.

The destination for **print** commands can also be a named data block. Data block names start with '$', see also **inline data (p. 47)**. When printing a string to a data block, embedded newline characters are expanded to generate multiple data block entries. This is a CHANGE.

Psdir

The **set psdir <directory>** command controls the search path used by the postscript terminal to find prologue.ps and character encoding files. You can use this mechanism to switch between different sets of locally-customized prolog files. The search order is

```
1) The directory specified by `set psdir`, if any
2) The directory specified by environmental variable GNUPLOT_PS_DIR
3) A built-in header or one from the default system directory
4) Directories set by `set loadpath`
```

Raxis

The commands **set raxis** and **unset raxis** toggle whether the polar axis is drawn separately from grid lines and the x axis. If the minimum of the current rrange is non-zero (and not autoscaled), then a white circle is drawn at the center of the polar plot to indicate that the plot lines and axes do not reach 0. The axis line is drawn using the same line type as the plot border. See **polar (p. 192)**, **rrange (p. 194)**, **rtics (p. 194)**, **rlabel (p. 194)**, **set grid (p. 153)**.

Rgbmax

Syntax:

```
set rgbmax {1.0 | 255}
unset rgbmax
```

The red/green/blue color components of an rgbimage plot are by default interpreted as integers in the range [0:255]. **set rgbmax 1.0** tells the program that data values used to generate the color components of a plot with **rgbimage** or **rgbalpha** are floating point values in the range [0:1]. **unset rgbmax** returns to the default integer range [0:255].

Rlabel

This command places a label above the r axis. The label will be drawn whether or not the plot is in polar mode. See **set xlabel (p. 214)** for additional keywords.

Rmargin

The command **set rmargin** sets the size of the right margin. Please see **set margin (p. 167)** for details.

Rrange

The **set rrange** command sets the range of the radial coordinate for a graph in polar mode. This has the effect of setting both xrange and yrange as well. The resulting xrange and yrange are both [-(rmax-rmin) : +(rmax-rmin)]. However if you later change the x or y range, for example by zooming, this does not change rrange, so data points continue to be clipped against rrange. Unlike other axes, autoscaling the raxis always results in rmin = 0. The **reverse** autoscaling flag is ignored. Note: Setting a negative value for rmin may produce unexpected results.

Rtics

The **set rtics** command places tics along the polar axis. The tics and labels are drawn to the right of the origin. The **mirror** keyword causes them to be drawn also to the left of the origin. See **polar (p. 192)**, **set xtics (p. 217)**, and **set mxtics (p. 173)** for discussion of keywords.

Samples

The default sampling rate of functions, or for interpolating data, may be changed by the **set samples** command. To change the sampling range for a particular plot, see **plot sampling (p. 121)**.

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 **plot smooth (p. 111)** re 2D data and **set cntrparam (p. 137)** and **set dgrid3d (p. 144)** re 3D data.

When a 2D 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 **set isosamples (p. 156)**.

Size

Syntax:

```
set size {{no}square | ratio <r> | noratio} {<xscale>,<yscale>}
show size
```

The <xscale> and <yscale> values are scale factors for the size of the plot, which includes the graph, labels, and margins.

Important note:

```
In earlier versions of gnuplot, some terminal types used the values from
'set size' to control also the size of the output canvas; others did not.
Almost all terminals now follow the following convention:
```

set term <terminal_type> size <XX>, <YY> controls the size of the output file, or **canvas**. Please see individual terminal documentation for allowed values of the size parameters. By default, the plot will fill this canvas.

set size <XX>, <YY> scales the plot itself relative to the size of the canvas. Scale values less than 1 will cause the plot to not fill the entire canvas. Scale values larger than 1 will cause only a portion of the plot to fit on the canvas. Please be aware that setting scale values larger than 1 may cause problems on some terminal types.

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. This is the 2D equivalent to the 3D command **set view equal xy**. 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).

set size square is a synonym for **set size 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 3D plots, but do affect 3D projections created using **set view map**. See also **set view equal (p. 211)**, which forces the x and y axes of a 3D onto the same scale.

Examples:

To set the size so that the plot fills the available canvas:

```
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
```

Spiderplot

The **set spiderplot** command switches interpretation of coordinates to a polar system in which each data point is mapped to a position along a radial axis. paxis 1 is always vertical; axes 2 to N proceed clockwise with even spacing. The command must be issued prior to plotting. It has additional effects equivalent to

```
set style data spiderplot
unset border
unset tics
set key noautotitle
set size ratio 1.0
```

Use **reset** to restore these after plotting.

Style

Default plotting styles are chosen with the **set style data** and **set style function** commands. See **plot with (p. 124)** for information about how to override the default plotting style for individual functions and data sets. See **plotting styles (p. 61)** or **plot with (p. 124)** 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 boxplot <boxplot style options>
set style circle radius <size> {clip|noclip}
set style ellipse size <size> units {xy|xx|yy} {clip|noclip}
set style fill <fillstyle>
set style histogram <histogram style options>
set style line <n> <linestyle>
set style rectangle <object options> <linestyle> <fillstyle>
set style textbox {<n>} {opaque|transparent} {{no}border} {fillcolor}
```

Set style arrow

Each terminal has a default set of arrow and point types, which can be seen by using the command **test**. **set style arrow** 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 | backhead | heads}
                        {size <length>,<angle>{,<backangle>} {fixed}}
                        {filled | empty | nofilled | noborder}
                        {front | back}
```

```
                              { {linestyle | ls <line_style>}
                                | {linetype | lt <line_type>}
                                  {linewidth | lw <line_width}
                                  {linecolor | lc <colorspec>}
                                  {dashtype | dt <dashtype>} }
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 modified using **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 (p. 32)** for details.

By default the size of the arrow head is reduced for very short arrows. This can be disabled using the **fixed** keyword after the **size** command.

<backangle> is the angle (in degrees) the back branches make with the arrow (in the same direction as **<angle>**). It is ignored if the style is **nofilled**.

Specifying **filled** produces filled arrow heads with a border line around the arrow head. Specifying **noborder** produces filled arrow heads with no border. In this case the tip of the arrow head lies exactly on the endpoint of the vector and the arrow head is slightly smaller overall. Dashed arrows should always use **noborder**, since a dashed border is ugly. Not all terminals support filled arrow heads.

The line style may be selected from a user-defined list of line styles (see **set style line (p. 200)**) 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 **set style arrow** 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 **set arrow (p. 129)** for further examples.

Boxplot

The **set style boxplot** command allows you to change the layout of plots created using the **boxplot** plot style.

Syntax:

```
set style boxplot {range <r> | fraction <f>}
                  {{no}outliers} {pointtype <p>}
```

```
                         {candlesticks | financebars}
                         {medianlinewidth <width>}
                         {separation <x>}
                         {labels off | auto | x | x2}
                         {sorted | unsorted}
```

The box in the boxplot always spans the range of values from the first quartile to the third quartile of the data points. The limit of the whiskers that extend from the box can be controlled in two different ways. By default the whiskers extend from each end of the box for a range equal to 1.5 times the interquartile range (i.e. the vertical height of the box proper). Each whisker is truncated back toward the median so that it terminates at a y value belonging to some point in the data set. Since there may be no point whose value is exactly 1.5 times the interquartile distance, the whisker may be shorter than its nominal range. This default corresponds to

```
set style boxplot range 1.5
```

Alternatively, you can specify the fraction of the total number of points that the whiskers should span. In this case the range is extended symmetrically from the median value until it encompasses the requested fraction of the data set. Here again each whisker is constrained to end at a point in the data set. To span 95% of the points in the set

```
set style boxplot fraction 0.95
```

Any points that lie outside the range of the whiskers are considered outliers. By default these are drawn as individual circles (pointtype 7). The option **nooutliers** disables this. If outliers are not drawn they do not contribute to autoscaling.

By default boxplots are drawn in a style similar to candlesticks, but you have the option of using instead a style similar to finance bars.

A crossbar indicating the median is drawn using the same line type as box boundary. If you want a thicker line for the median

```
set style boxplot medianlinewidth 2.0
```

If you want no median line, set this to 0.

If the using specification for a boxplot contains a fourth column, the values in that column will be interpreted as the discrete leveles of a factor variable. In this case more than one boxplots may be drawn, as many as the number of levels of the factor variable. These boxplots will be drawn next to each other, the distance between them is 1.0 by default (in x-axis units). This distance can be changed by the option **separation**.

The **labels** option governs how and where these boxplots (each representing a part of the dataset) are labeled. By default the value of the factor is put as a tick label on the horizontal axis – x or x2, depending on which one is used for the plot itself. This setting corresponds to option **labels auto**. The labels can be forced to use either of the x or x2 axes – options **labels x** and **labels x2**, respectively –, or they can be turned off altogether with the option **labels off**.

By default the boxplots corresponding to different levels of the factor variable are not sorted; they will be drawn in the same order the levels are encountered in the data file. This behavior corresponds to the **unsorted** option. If the **sorted** option is active, the levels are first sorted alphabetically, and the boxplots are drawn in the sorted order.

The **separation**, **labels**, **sorted** and **unsorted** option only have an effect if a fourth column is given the plot specification.

See **boxplot (p. 63)**, **candlesticks (p. 65)**, **financebars (p. 69)**.

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 (p. 61)** for the choices. **show style data** shows the current default data plotting style.

Set style fill

The **set style fill** command is used to set the default style of the plot elements in plots with boxes, histograms, candlesticks and filledcurves. This default can be superseded by fillstyles attached to individual plots.

Note that there is a separate default fill style for rectangles created by **set obj**. See **set style rectangle (p. 202)**.

Syntax:

```
set style fill {empty
                | {transparent} solid {<density>}
                | {transparent} pattern {<n>}}
               {border {lt} {lc <colorspec>} | noborder}
```

The **empty** option causes filled areas not to be filled. This is the default.

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.

Fill color (**fillcolor <colorspec>**) is distinct from fill style. I.e. plot elements or objects can share a fillstyle while retaining separate colors. In most places where a fillstyle is accepted you can also specify a fill color. Fillcolor may be abbreviated **fc**. Otherwise the fill color is take from the current linetype. Example:

```
plot FOO with boxes fillstyle solid 1.0 fillcolor "cyan"
```

Set style fill border The bare keyword **border** causes the filled object to be surrounded by a solid line of the current linetype and color. You can change the color of this line by adding either a linetype or a linecolor. **noborder** specifies that no bounding line is drawn. Examples:

```
# Half-intensity fill, full intensity border in same color
set style fill solid 0.5 border
# Half-transparent fill, solid black border (linetype -1)
set style fill transparent solid 0.5 border -1
# Pattern fill in current color, border using color of linetype 5
plot ... with boxes fillstyle pattern 2 border lt 5
# Fill area in cyan, border in blue
plot ... with boxes fillcolor "cyan" fs solid border linecolor "blue"
```

Note: The border property of a fill style only affects plots drawn **with filledcurves** in the default mode (closed curve).

Set style fill transparent Some terminals support the attribute **transparent** for filled areas. In the case of transparent solid fill areas, the **density** parameter is interpreted as an alpha value; that is, density 0 is fully transparent, density 1 is fully opaque. In the case of transparent pattern fill, the background of the pattern is either fully transparent or fully opaque.

Note that there may be additional limitations on the creation or viewing of graphs containing transparent fill areas. For example, the png terminal can only use transparent fill if the "truecolor" option is set. Some pdf viewers may not correctly display the fill areas even if they are correctly described in the pdf file. Ghostscript/gv does not correctly display pattern-fill areas even though actual PostScript printers generally have no problem.

Set style function

The **set style function** command changes the default plotting style for function plots (e.g. lines, points, filledcurves). See **plotting styles (p. 61)**.

Syntax:

```
set style function <plotting-style>
show style function
```

Set style increment

Note: This command has been deprecated. Instead please use the newer command **set linetype**, which redefines the linetypes themselves rather than searching for a suitable temporary line style to substitute. See **set linetype (p. 164)**

Syntax:

```
set style increment {default|userstyles}
show style increment
```

By default, successive plots within the same graph will use successive linetypes from the default set for the current terminal type. However, choosing **set style increment user** allows you to step through the user-defined line styles rather than through the default linetypes.

Example:

```
set style line 1 lw 2 lc rgb "gold"
set style line 2 lw 2 lc rgb "purple"
set style line 4 lw 1 lc rgb "sea-green"
set style increment user

plot f1(x), f2(x), f3(x), f4(x)
```

should plot functions f1, f2, f4 in your 3 newly defined line styles. If a user-defined line style is not found then the corresponding default linetype is used instead. E.g. in the example above, f3(x) will be plotted using the default linetype 3.

Set style line

Each terminal has a default set of line and point types, which can be seen by using the command **test**. **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> {{linetype  | lt} <line_type> | <colorspec>}
                       {{linecolor | lc} <colorspec>}
                       {{linewidth | lw} <line_width>}
                       {{pointtype | pt} <point_type>}
                       {{pointsize | ps} <point_size>}
```

```
                              {{pointinterval | pi} <interval>}
                              {{pointnumber | pn} <max_symbols>}
                              {{dashtype | dt} <dashtype>}
                              {palette}
unset style line
show style line
```

default sets all line style parameters to those of the linetype with that same index.

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.

Line styles created by this mechanism do not replace the default linetype styles; both may be used. Line styles are temporary. They are lost whenever you execute a **reset** command. To redefine the linetype itself, please see **set linetype (p. 164)**.

The line and point types default to the index value. The exact symbol that is drawn for that index value may vary from one terminal type to another.

The line width and point size are multipliers for the current terminal's default width and size (but note that <point_size> here is unaffected by the multiplier given by the command**set pointsize**).

The **pointinterval** controls the spacing between points in a plot drawn with style **linespoints**. The default is 0 (every point is drawn). For example, **set style line N pi 3** defines a linestyle that uses pointtype N, pointsize and linewidth equal to the current defaults for the terminal, and will draw every 3rd point in plots using **with linespoints**. A negative value for the interval is treated the same as a positive value, except that some terminals will try to interrupt the line where it passes through the point symbol.

The **pointnumber** property is similar to **pointinterval** except that rather than plotting every Nth point it limits the total number of points to N.

Not all terminals support the **linewidth** and **pointsize** 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 (p. 48)**, **colorspec (p. 49)**, **set palette (p. 180)**, **colornames (p. 139)**, **cbrange (p. 225)**.

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 **palette** 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 **palette**. Note, that this works only on some terminals. See also **set palette (p. 180)**, **set pm3d (p. 187)**.

```
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.

Set style circle

Syntax:

```
set style circle {radius {graph|screen} <R>}
                 {{no}wedge}
                 {clip|noclip}
```

This command sets the default radius used in plot style "with circles". It applies to data plots with only 2 columns of data (x,y) and to function plots. The default is "set style circle radius graph 0.02". **Nowedge** disables drawing of the two radii that connect the ends of an arc to the center. The default is **wedge**. This parameter has no effect on full circles. **Clip** clips the circle at the plot boundaries, **noclip** disables this. Default is **clip**.

Set style rectangle

Rectangles defined with the **set object** command can have individual styles. However, if the object is not assigned a private style then it inherits a default that is taken from the **set style rectangle** command.

Syntax:

```
set style rectangle {front|back} {lw|linewidth <lw>}
                    {fillcolor <colorspec>} {fs <fillstyle>}
```

See **colorspec (p. 49)** and **fillstyle (p. 199)**. **fillcolor** may be abbreviated as **fc**.

Examples:

```
set style rectangle back fc rgb "white" fs solid 1.0 border lt -1
set style rectangle fc linsestyle 3 fs pattern 2 noborder
```

The default values correspond to solid fill with the background color and a black border.

Set style ellipse

Syntax:

```
set style ellipse {units xx|xy|yy}
                  {size {graph|screen} <a>, {{graph|screen} <b>}}
                  {angle <angle>}
                  {clip|noclip}
```

This command governs whether the diameters of ellipses are interpreted in the same units or not. Default is **xy**, which means that the major diameter (first axis) of ellipses will be interpreted in the same units as the x (or x2) axis, while the minor (second) diameter in those of the y (or y2) axis. In this mode the ratio of the ellipse axes depends on the scales of the plot axes and aspect ratio of the plot. When set to **xx** or **yy**, both axes of all ellipses will be interpreted in the same units. This means that the ratio of the axes of the plotted ellipses will be correct even after rotation, but either their vertical or horizontal extent will not be correct.

This is a global setting that affects all ellipses, both those defined as objects and those generated with the **plot** command, however, the value of **units** can also be redefined on a per-plot and per-object basis.

It is also possible to set a default size for ellipses with the **size** keyword. This default size applies to data plots with only 2 columns of data (x,y) and to function plots. The two values are interpreted as the major and minor diameters (as opposed to semi-major and semi-minor axes) of the ellipse.

The default is "set style ellipse size graph 0.05,0.03".

Last, but not least it is possible to set the default orientation with the **angle** keyword. The orientation, which is defined as the angle between the major axis of the ellipse and the plot's x axis, must be given in degrees.

Clip clips the ellipse at the plot boundaries, **noclip** disables this. Default is **clip**.

For defining ellipse objects, see **set object ellipse (p. 176)**; for the 2D plot style, see **ellipses (p. 67)**.

Set style parallelaxis

Syntax:

```
set style parallelaxis {front|back} {line-properties}
```

Determines the line type and layer for drawing the vertical axes in plots **with parallelaxes**. See **with parallelaxes (p. 76)**, **set paxis (p. 186)**.

Set style spiderplot

Syntax:

```
set style spiderplot
          {fillstyle <fillstyle-properties>}
          {<line-properties> | <point-properties>}
```

This commands controls the default appearance of spider plots. The fill, line, and point properties can be modified in the first component of the plot command. The overall appearance of the plot is also affected by other settings such as **set grid spiderplot**. See also **set paxis (p. 186)**, **spiderplot (p. 79)**. Example:

```
# Default spider plot will be a polygon with a thick border but no fill
set style spiderplot fillstyle empty border lw 3
# This one will additionally place an open circle (pt 6) at each axis
plot for [i=1:6] DATA pointtype 6 pointsize 3
```

Set style textbox

Syntax:

```
set style textbox {<boxstyle-index>}
          {opaque|transparent} {fillcolor <color>}
          {{no}border {<bordercolor>}}{linewidth <lw>}
          {margins <xmargin>,<ymargin>}
```

This command controls the appearance of labels with the attribute 'boxed'. Terminal types that do not support boxed text will ignore this style. Note: Implementation for some terminals (svg, latex) is incomplete. Most terminals cannot place a box correctly around rotated text.

Three numbered textbox styles can be defined. If no boxstyle index <bs> is given, the default (unnumbered) style is changed. Example:

```
# default style has only a black border
set style textbox transparent border lc "black"
```

```
# style 2 (bs 2) has a light blue background with no border
set style textbox 2 opaque fc "light-cyan" noborder
set label 1 "I'm in a box" boxed
set label 2 "I'm blue" boxed bs 2
```

Surface

The **set surface** command is only relevant for 3D plots (**splot**).

Syntax:

```
set surface {implicit|explicit}
unset surface
show surface
```

unset surface will cause **splot** to not draw points or lines corresponding to any of the function or data file points. This is mainly useful for drawing only contour lines rather than the surface they were derived from. Contours may still be drawn on the surface, depending on the **set contour** option. To turn off the surface for an individual function or data file while leaving others active, use the **nosurface** keyword in the **splot** command. The combination **unset surface; set contour base** is useful for displaying contours on the grid base. See also **set contour (p. 140)**.

If a 3D data set is recognizable as a mesh (grid) then by default the program implicitly treats the plot style **with lines** as requesting a gridded surface. See **grid_data (p. 229)**. The command **set surface explicit** suppresses this expansion, plotting only the individual lines described by separate blocks of data in the input file. A gridded surface can still be plotted by explicitly requesting splot **with surface**.

Table

When **table** mode is enabled, **plot** and **splot** commands print out a multicolumn text table of values

```
X Y {Z} <flag>
```

rather than creating an actual plot on the current terminal. The flag character is "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 tickmarks (see **set format (p. 149)**), 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. The same method can be used to save interpolated data (see **set samples (p. 194)** and **set dgrid3d (p. 144)**).

Syntax:

```
set table {"outfile" | $datablock} {append}
          {separator {whitespace|tab|comma|"<char>"}}
plot <whatever>
unset table
```

Subsequent tabular output is written to "outfile", if specified, otherwise it is written to stdout or other current value of **set output**. If **outfile** exists it will be replaced unless the **append** keyword is given. Alternatively, tabular output can be redirected to a named data block. Data block names start with '$', see also **inline data (p. 47)**. You must explicitly **unset table** in order to go back to normal plotting on the current terminal.

The **separator** character can be used to output csv (comma separated value) files. This mode only affects plot style **with table**. See **plot with table (p. 204)**.

Plot with table

This discussion applies only to the special plot style **with table**.

To avoid any style-dependent processing of the input data being tabulated (smoothing, errorbar expansion, secondary range checking, etc), or to increase the number of columns that can be tabulated, use the keyword "table" instead of a normal plot style. In this case the output does not contain an extra, last, column of flags **i**, **o**, **u** indicated inrange/outrange/undefined. The destination for output must first be specified with **set table <where>**. For example

```
set table $DATABLOCK1
plot <file> using 1:2:3:4:($5+$6):(func($7)):8:9:10 with table
```

Because there is no actual plot style in this case the columns do not correspond to specific axes. Therefore xrange, yrange, etc are ignored.

If a **using** term evaluates to a string, the string is tabulated. Numerical data is always written with format %g. If you want some other format use sprintf or gprintf to create a formatted string.

```
plot <file> using ("File 1"):1:2:3 with table
plot <file> using (sprintf("%4.2f",$1)) : (sprintf("%4.2f",$3)) with table
```

To create a csv file use

```
set table "tab.csv" separator comma
plot <foo> using 1:2:3:4 with table
```

[EXPERIMENTAL] To select only a subset of the data points for tabulation you can provide an input filter condition (**if <expression>**) at the end of the command. Note that the input filter may reference data columns that are not part of the output. This feature may change substantially before appearing in a released version of gnuplot.

```
plot <file> using 1:2:($4+$5) with table if (strcol(3) eq "Red")
plot <file> using 1:2:($4+$5) with table if (10. < $1 && $1 < 100.)
plot <file> using 1:2:($4+$5) with table if (filter($6,$7) != 0)
```

Terminal

gnuplot supports many different graphics devices. Use **set terminal** to tell **gnuplot** what kind of output to generate. Use **set output** 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 **set output** are used together, it is safest to give **set terminal** first, because some terminals set a flag which is needed in some operating systems.

Some terminals have many additional options. 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 more information, see the **complete list of terminals (p. 236)**.

Termoption

The **set termoption** 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>}"
set termoption fontscale <scale>
set termoption {linewidth <lw>}{lw <lw>}
```

Theta

Polar coordinate plots are by default oriented such that theta = 0 is on the right side of the plot, with theta increasing as you proceed counterclockwise so that theta = 90 degrees is at the top. **set theta** allows you to change the origin and direction of the polar angular coordinate theta.

```
set theta {right|top|left|bottom}
set theta {clockwise|cw|counterclockwise|ccw}
```

unset theta restores the default state "set theta right ccw".

Tics

The **set tics** command controls the tic marks and labels on all axes at once.

The tics may be turned off with the **unset tics** command, and may be turned on (the default state) with **set tics**. Fine control of tics on individual axes is possible using the alternative commands **set xtics**, **set ztics**, etc.

Syntax:

```
set tics {axis | border} {{no}mirror}
         {in | out} {front | back}
         {{no}rotate {by <ang>}} {offset <offset> | nooffset}
         {left | right | center | autojustify}
         {format "formatstring"} {font "name{,<size>}"} {{no}enhanced}
         { textcolor <colorspec> }
set tics scale {default | <major> {,<minor>}}
unset tics
show tics
```

The options can be applied to a single axis (x, y, z, x2, y2, cb), e.g.

```
set xtics rotate by -90
unset cbtics
```

All tic marks are drawn using the same line properties as the plot border (see **set border (p. 133)**).

Set tics **back** or **front** applies to all axes at once, but only for 2D plots (not splot). It controls whether the tics are placed behind or in front of the plot elements, in the case that there is overlap.

axis or **border** 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 **set border (p. 133)**). The relevant margin settings will usually be sized badly by the automatic layout algorithm in this case.

mirror tells **gnuplot** to put unlabeled 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.

set tics scale controls the size of the tic marks. The first value <major> controls the auto-generated or user-specified major tics (level 0). The second value controls the auto-generated or user-specified minor tics (level 1). <major> defaults to 1.0, <minor> defaults to <major>/2. Additional values control the size of user-specified tics with level 2, 3, ... Default tic sizes are restored by **set tics 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 (p. 32)** for details. **nooffset** switches off the offset.

By default, tic labels are justified automatically depending on the axis and rotation angle to produce aesthetically pleasing results. If this is not desired, justification can be overridden with an explicit **left**, **right** or **center** keyword. **autojustify** restores the default behavior.

set tics with no options restores mirrored, inward-facing tic marks for the primary axes. All other settings are retained.

See also **set xtics (p. 217)** for more control of major (labeled) tic marks and **set mxtics** for control of minor tic marks. These commands provide control of each axis independently.

Ticslevel

Deprecated. See **set xyplane (p. 221)**.

Ticscale

The **set ticscale** command is deprecated, use **set tics scale** instead.

Timestamp

The command **set timestamp** places the current time and date in the plot margin.

Syntax:
```
set timestamp {"<format>"} {top|bottom} {{no}rotate}
              {offset <xoff>{,<yoff>}} {font "<fontspec>"}
              {textcolor <colorspec>}
unset timestamp
show timestamp
```

The format string is 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 along the top left or bottom left margin (default: bottom). **rotate** writes the timestamp vertically. 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 **timestamp**.

Example:
```
set timestamp "%d/%m/%y %H:%M" offset 80,-2 font "Helvetica"
```

See **set timefmt (p. 208)** for more information about time format strings.

Timefmt

This command sets the default format used to input time data. See **set xdata time (p. 213)**, **timecolumn (p. 39)**.

Syntax:

```
set timefmt "<format string>"
show timefmt
```

The valid formats for both **timefmt** and **timecolumn** are:

Time Series timedata Format Specifiers	
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, integer 0–60 on output, (double) on input
`%b`	three-character abbreviation of the name of the month
`%B`	name of the month
`%p`	two character match to one of: am AM pm PM

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. If a decimal point immediately follows the field read by %S, the decimal and any following digits are interpreted as a fractional second. %Y reads four digits. %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 **using** specification if your file contains timedata.

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. **gnuplot** will determine the proper month and weekday from the numerical values. See **set format (p. 149)** for more details about these and other options for printing time data.

When reading two-digit years with %y, values 69-99 refer to the 20th century, while values 00-68 refer to the 21st century. NB: This is in accordance with the UNIX98 spec, but conventions vary widely and two-digit year values are inherently ambiguous.

If the %p format returns "am" or "AM", hour 12 will be interpreted as hour 0. If the %p format returns "pm" or "PM", hours $<$ 12 will be increased by 12.

See also **set xdata (p. 213)** and **time/date (p. 60)** 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.`

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 **set label**.

Syntax:
```
set title {"<title-text>"} {offset <offset>} {font "<font>{,<size>}"}
          {{textcolor | tc} {<colorspec> | default}} {{no}enhanced}
show title
```

If <offset> is specified by either x,y or x,y,z the title is moved by the given offset. It may be preceded by **first**, **second**, **graph**, **screen**, or **character** to select the coordinate system. See **coordinates (p. 32)** for details. By default, the **character** coordinate system is used. For example, "**set title offset 0,-1**" will change only the y offset of the title, moving the title down by roughly the height of one character. The size of a character depends on both the font and the terminal.

<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 <colorspec> changes the color of the text. <colorspec> can be a linetype, an rgb color, or a palette mapping. See help for **colorspec (p. 49)** and **palette (p. 39)**.

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 (p. 59)** for details about the processing of backslash sequences and the distinction between single- and double-quotes.

Tmargin

The command **set tmargin** sets the size of the top margin. Please see **set margin (p. 167)** for details.

Trange

Syntax: set trange [tmin:tmax] The range of the parametric variable t is useful in three contexts. 1) In parametric mode **plot** commands it limits the range of sampling
```
for both generating functions.  See ‘set parametric‘, ‘set samples‘.
```

2) In polar mode **plot** commands it limits or defines the range of the
```
angular parameter theta. See ‘polar‘.
```

3) In **plot** or **splot** commands using 1-dimensional sampled data via
```
the pseudofile "+".  See ‘sampling 1D‘, ‘special-filenames‘.
```

Ttics

The **set ttics** command places tics around the perimeter of a polar plot. This is the border if **set border polar** is enabled, otherwise the outermost circle of the polar grid drawn at the rightmost ticmark along the r axis. See **set grid (p. 153)**, **set rtics (p. 194)**. The angular position is always labeled in degrees. The full perimeter can be labeled regardless of the current trange setting. The desired range of the tic labels should be given as shown below. Additional properties of the tic marks can be set. See **xtics (p. 217)**.
```
set ttics -180, 30, 180
set ttics add ("Theta = 0" 0)
set ttics font ":Italic" rotate
```

Urange

Syntax: set urange [umin:umax] The range of the parametric variables u and v is useful in two contexts. 1) **splot** in parametric mode. See **set parametric (p. 185)**, **set isosamples (p. 156)**. 2) generating 2-dimension sampled data for either **plot** or **splot** using the pseudofile "++". See **sampling 2D (p. 121)**.

Variables

The **show variables** command lists the current value of user-defined and internal variables. Gnuplot internally defines variables whose names begin with GPVAL_, MOUSE_, FIT_, and TERM_.

Syntax:

```
show variables      # show variables that do not begin with GPVAL_
show variables all  # show all variables including those beginning GPVAL_
show variables NAME # show only variables beginning with NAME
```

Version

The **show version** 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.

Vgrid

Syntax:

```
set vgrid $gridname {size N}
unset vgrid $gridname
show vgrid
```

If the named grid already exists, mark it as active (use it for subsequent **vfill** and **voxel** operations). If a new size is given, replace the existing content with a zero-filled N x N x N grid. If a grid with this name does not already exist, allocate an N x N x N grid (default N=100), zero the contents, and mark it as active. Note that grid names must begin with '$'.

show vgrid lists all currently defined voxel grids. Example output:

```
$vgrid1: (active)
         size 100 X 100 X 100
         vxrange [-4:4]  vyrange[-4:4]  vzrange[-4:4]
         non-zero voxel values:  min 0.061237 max 94.5604
         number of zero voxels:  992070   (99.21%)
```

unset vgrid $gridname releases all data structures associated with that voxel grid. The data structures are also released by **reset session**. The function **voxel(x,y,z)** returns the value of the active grid point nearest that coordinate. See also **splot voxel-grids (p. 230)**.

View

The **set view** command sets the viewing angle for **splot**s. It controls how the 3D coordinates of the plot are mapped into the 2D 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>}}}
set view map {scale <scale>}
set view projection {xy|xz|yz}
set view {no}equal {xy|xyz}
set view azimuth <angle>
show view
```

where <rot_x> and <rot_z> control the rotation angles (in degrees) in a virtual 3D 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 is useful for **contour** plots or 2D heatmaps using pm3d mode rather than **with image**. In the latter case, take care that you properly use **zrange** and **cbrange** 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.

Azimuth

```
set view azimuth <angle-in-degrees>
```

The setting of azimuth affects the orientation of the z axis in a 3D graph (splot). At the default azimuth = 0 the z axis of the plot lies in the plane orthogonal to the screen horizontal. I.e. the projection of the z axis lies along the screen vertical. Non-zero azimuth rotates the plot about the line of sight through the origin so that a projection of the z axis is no longer vertical. When azimuth = 90 the z axis is horizontal rather than vertical.

Equal_axes

The command **set view equal xy** forces the unit length of the x and y axes to be on the same scale, and chooses that scale so that the plot will fit on the page. The command **set view equal xyz** additionally sets the z axis scale to match the x and y axes; however there is no guarantee that the current z axis range will fit within the plot boundary. By default all three axes are scaled independently to fill the available area.

See also **set xyplane (p. 221)**.

Projection

Syntax:

```
set view projection {xy|xz|yz}
```

Rotates the view angles of a 3D plot so that one of the primary planes xy, xz, or yz lies in the plane of the plot. Axis labels and tics positioning is adjusted accordingly; tics and labels on the third axis are disabled. The plot is scaled up to approximately match the size that 'plot' would generate for the same axis ranges. **set view projection xy** is equivalent to **set view map**.

Vrange

Syntax: set vrange [vmin:vmax] The range of the parametric variables u and v is useful in two contexts. 1) **splot** in parametric mode. See **set parametric (p. 185)**, **set isosamples (p. 156)**. 2) generating 2-dimension sampled data for either **plot** or **splot** using the pseudofile "++". See **sampling 2D (p. 121)**.

Vxrange

Syntax: set vxrange [vxmin:vxmax]

Establishes the range of x coordinates spanned by the active voxel grid. Analogous commands **set vyrange** and **set vzrange** exist for the other two dimensions of the voxel grid. If no explicit ranges have been set prior to the first **vclear**, **vfill**, or **voxel(x,y,z) =** command, vmin and vmax will be copied from the current values of **xrange**.

Vyrange

See **set vxrange (p. 212)**

Vzrange

See **set vxrange (p. 212)**

Walls

Syntax:

```
set walls
set wall {x0|y0|z0|x1|y1} {<fillstyle>} {fc <fillcolor>}
```

3D surfaces drawn by **splot (p. 225)** lie within a normalized unit cube regardless of the x y and z axis ranges. The bounding walls of this cube are described by the planes (graph coord x == 0), (graph coord x == 1), etc. The **set walls** command renders the walls x0 y0 and z0 as solid surfaces. By default these surfaces are semi-transparent (fillstyle transparent solid 0.5). You can customize which walls are drawn and also their individual color and fill style. If you choose to enable walls, you may also want to use **set xyplane 0**. Example:

```
set wall z0 fillstyle solid 1.0 fillcolor "gray"
```

X2data

The **set x2data** command sets data on the x2 (top) axis to timeseries (dates/times). Please see **set xdata (p. 213)**.

X2dtics

The **set x2dtics** command changes tics on the x2 (top) axis to days of the week. Please see **set xdtics (p. 214)** for details.

X2label

The **set x2label** command sets the label for the x2 (top) axis. Please see **set xlabel (p. 214)**.

X2mtics

The **set x2mtics** command changes tics on the x2 (top) axis to months of the year. Please see **set xmtics (p. 215)** for details.

X2range

The **set x2range** command sets the horizontal range that will be displayed on the x2 (top) axis. See **set xrange (p. 215)** for the full set of command options. See also **set link (p. 165)**.

X2tics

The **set x2tics** command controls major (labeled) tics on the x2 (top) axis. Please see **set xtics (p. 217)** for details.

X2zeroaxis

The **set x2zeroaxis** command draws a line at the origin of the x2 (top) axis (y2 = 0). For details, please see **set zeroaxis (p. 224)**.

Xdata

This command controls interpretation of data on the x axis. An analogous command acts on each of the other axes.

Syntax:

```
set xdata time
show xdata
```

The same syntax applies to **ydata**, **zdata**, **x2data**, **y2data** and **cbdata**.

The **time** option signals that data represents a time/date in seconds. The current version of gnuplot stores time to a millisecond precision.

If no option is specified, the data interpretation reverts to normal.

Time

set xdata time indicates that the x coordinate represents a date or time to millisecond precision. There is an analogous command **set ydata time**.

There are separate format mechanisms for interpretation of time data on input and output. Input data is read from a file either by using the global **timefmt** or by using the function timecolumn() as part of the plot command. These input mechanisms also apply to using time values to set an axis range. See **set timefmt (p. 208)**, **timecolumn (p. 39)**.

Example:

```
set xdata time
set timefmt "%d-%b-%Y"
set xrange ["01-Jan-2013" : "31-Dec-2014"]
plot DATA using 1:2
```

or

```
plot DATA using (timecolumn(1,"%d-%b-%Y")):2
```

For output, i.e. tick labels along that axis or coordinates output by mousing, the function 'strftime' (type "man strftime" on unix to look it up) is used to convert from the internal time in seconds to a string representation of a date. **gnuplot** tries to figure out a reasonable format for this. You can customize the format using either **set format x** or **set xtics format**. See **time_specifiers (p. 151)** for a special set of time format specifiers. See also **time/date (p. 60)** for more information.

Xdtics

The **set xdtics** 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 **ydtics**, **zdtics**, **x2dtics**, **y2dtics** and **cbdtics**.

See also the **set format (p. 149)** command.

Xlabel

The **set xlabel** command sets the x axis label. Similar commands set labels on the other axes.

Syntax:

```
set xlabel {"<label>"} {offset <offset>} {font "<font>{,<size>}"}
           {textcolor <colorspec>} {{no}enhanced}
           {rotate by <degrees> | rotate parallel | norotate}
show xlabel
```

The same syntax applies to **x2label**, **ylabel**, **y2label**, **zlabel** and **cblabel**.

If <offset> is specified by either x,y or x,y,z the label is moved by the given offset. It may be preceded by **first**, **second**, **graph**, **screen**, or **character** to select the coordinate system. See **coordinates (p. 32)** 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.

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., **"set y2label"**.

The default positions of the axis labels are as follows:

xlabel: The x-axis label is centered below the bottom of the plot.

ylabel: The y-axis label is centered to the left of the plot, defaulting to either horizontal or vertical orientation depending on the terminal type. The program may not reserve enough space to the left of the plot to hold long non-rotated ylabel text. You can adjust this with **set lmargin**.

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 to the 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 plot but below the 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 orientation (rotation angle) of the x, x2, y and y2 axis labels in 2D plots can be changed by specifying **rotate by <degrees>**. The orientation of the x and y axis labels in 3D plots defaults to horizontal but can be changed to run parallel to the axis by specifying **rotate parallel**.

If you are not satisfied with the default position of an axis label, use **set label** instead–that command gives you much more control over where text is placed.

Please see **syntax (p. 59)** for further information about backslash processing and the difference between single- and double-quoted strings.

Xmtics

The **set xmtics** 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 **unset xmtics**. Similar commands perform the same duties for the other axes.

Syntax:

```
set xmtics
unset xmtics
show xmtics
```

The same syntax applies to **x2mtics**, **ymtics**, **y2mtics**, **zmtics** and **cbmtics**.

See also the **set format (p. 149)** command.

Xrange

The **set xrange** 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} {{no}extend}
           | 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 **set timefmt** format. If <min> or <max> is omitted the current value will not be changed. See below for full autoscaling syntax. See also **noextend (p. 132)**.

The same syntax applies to **yrange**, **zrange**, **x2range**, **y2range**, **cbrange**, **rrange**, **trange**, **urange** and **vrange**.

See **set link (p. 165)** for options that link the ranges of x and x2, or y and y2.

The **reverse** option reverses the direction of an autoscaled axis. For example, if the data values range from 10 to 100, it will autoscale to the equivalent of set xrange [100:10]. The **reverse** flag has no effect if the axis is not autoscaled. NB: This is a change introduced in version 4.7.

Autoscaling: If <min> (the same applies for correspondingly to <max>) is an asterisk "*" autoscaling is turned on. The range in which autoscaling is being performed may be limited by a lower bound <lb> or an upper bound <ub> or both. The syntax is

```
{ <lb> < } * { < <ub> }
```

For example,

```
0 < * < 200
```

sets <lb> = 0 and <ub> = 200. With such a setting <min> would be autoscaled, but its final value will be between 0 and 200 (both inclusive despite the '<' sign). If no lower or upper bound is specified, the '<' to also be omitted. If <ub> is lower than <lb> the constraints will be turned off and full autoscaling will happen. This feature is useful to plot measured data with autoscaling but providing a limit on the range, to clip outliers, or to guarantee a minimum range that will be displayed even if the data would not need such a big range.

The **writeback** option essentially saves the range found by **autoscale** in the buffers that would be filled by **set xrange**. 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 **show yrange** after each command in the above example should help you understand what is going on.

In 2D, **xrange** and **yrange** determine the extent of the axes, **trange** determines the range of the parametric variable in parametric mode or the range of the angle in polar mode. Similarly in parametric 3D, **xrange**, **yrange**, and **zrange** govern the axes and **urange** and **vrange** govern the parametric variables.

In polar mode, **rrange** 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. **xrange** and **yrange** 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

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 [*:]
```

To autoscale xmin but keeping xmin positive:

```
set xrange [0<*:]
```

To autoscale x but keep minimum range of 10 to 50 (actual might be larger):

```
set xrange [*<10:50<*]
```

Autoscaling but limit maximum xrange to -1000 to 1000, i.e. autoscaling within [-1000:1000]

```
set xrange [-1000<*:*<1000]
```

Make sure xmin is somewhere between -200 and 100:

```
set xrange [-200<*<100:]
```

Extend

set xrange noextend is the same as **set autoscale x noextend**. See **noextend (p. 132)**.

Xtics

Fine control of the major (labeled) tics on the x axis is possible with the **set xtics** command. The tics may be turned off with the **unset xtics** command, and may be turned on (the default state) with **set xtics**. 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}
          {left | right | center | autojustify}
          {add}
          {  autofreq
           | <incr>
           | <start>, <incr> {,<end>}
           | ({"<label>"} <pos> {<level>} {,{"<label>"}...) }
          {format "formatstring"} {font "name{,<size>}"} {{no}enhanced}
          { numeric | timedate | geographic }
          {{no}logscale}
          { rangelimited }
          { textcolor <colorspec> }
unset xtics
show xtics
```

The same syntax applies to **ytics**, **ztics**, **x2tics**, **y2tics** and **cbtics**.

axis or **border** 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 unlabeled 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 **set border**.

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 (p. 32)** for details. **nooffset** switches off the offset.

Example:

Move xtics more closely to the plot.

```
set xtics offset 0,graph 0.05
```

By default, tic labels are justified automatically depending on the axis and rotation angle to produce aesthetically pleasing results. If this is not desired, justification can be overridden with an explicit **left**, **right** or **center** keyword. **autojustify** restores the default behavior.

set xtics 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.

Tic positions are calculated automatically by default or if the **autofreq** option is given.

A series of tic positions can be specified by giving either a tic interval alone, or a start point, interval, and end point (see **xtics series (p. 218)**).

Individual tic positions can be specified individually by providing an explicit list of positions, where each position may have an associated text label. See **xtics list (p. 219)**.

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 (unlabeled) tics can be added automatically by the **set mxtics** command, or at explicit positions by the **set xtics ("" <pos> 1, ...)** form.

The appearance of the tics (line style, line width etc.) is determined by the border line (see **set border (p. 133)**), even if the tics are drawn at the axes.

Xtics series

Syntax:

```
set xtics <incr>
set xtics <start>, <incr>, <end>
```

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
```

Xtics list

Syntax:
```
set xtics {add} ("label1" <pos1> <level1>, "label2" <pos2> <level2>, ...)
```

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.

The label is a string enclosed by quotes or a string-valued expression. It may contain formatting information for converting the position into its label, such as "%3f clients", or it may be the empty string "". See **set format (p. 149)** 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. Level 1 generates a minor tic. Labels are never printed for minor tics. Major and minor tics may be auto-generated by the program or specified explicitly by the user. Tics with level 2 and higher must be explicitly specified by the user, and take priority over auto-generated tics. The size of tics marks at each level is controlled by the command **set tics scale**.

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 labeled. In the third, only the end tics are labeled. 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.

Xtics timedata

Times and dates are stored internally as a number of seconds.

Input: Non-numeric time and date values are converted to seconds on input using the format specifier in **timefmt**. Axis positions and range limits also may be given as quoted dates or times interpreted using **timefmt**. If the <start>, <incr>, <end> form is used, <incr> must be in seconds. Use of **timefmt** to interpret input data, range, and tic positions is triggered by **set xdata time**.

Output: Axis tic labels are generated using a separate format specified either by **set format** or **set xtics format**. By default the usual numeric format specifiers are expected (**set xtics numeric**). Other options are geographic coordinates (**set xtics geographic**), or times or dates (**set xtics time**).

Note: For backward compatibility with earlier gnuplot versions, the command **set xdata time** will implicitly also do **set xtics time**, and **set xdata** or **unset xdata** will implicitly reset to **set xtics numeric**. However you can change this with a later call to **set xtics**.

Examples:

```
set xdata time           # controls interpretation of input data
set timefmt "%d/%m"      # format used to read input data
set xtics timedate       # controls interpretation of output format
set xtics format "%b %d" # format used for tic labels
set xrange ["01/12":"06/12"]
set xtics "01/12", 172800, "05/12"

set xdata time
set timefmt "%d/%m"
set xtics format "%b %d" time
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 unlabeled.

Geographic

set xtics geographic indicates that x-axis values are to be interpreted as a geographic coordinate measured in degrees. Use **set xtics format** or **set format x** to specify the appearance of the axis tick labels. The format specifiers for geographic data are as follows:

```
%D                     = integer degrees
%<width.precision>d  = floating point degrees
%M                     = integer minutes
%<width.precision>m  = floating point minutes
%S                     = integer seconds
%<width.precision>s  = floating point seconds
%E                     = label with E/W instead of +/-
%N                     = label with N/S instead of +/-
```

For example, the command **set format x "%Ddeg %5.2mmin %E"** will cause x coordinate -1.51 to be labeled as **" 1deg 30.60min W"**.

If the xtics are left in the default state (**set xtics numeric**) the coordinate will be reported as a decimal number of degrees, and **format** will be assumed to contain normal numeric format specifiers rather than the special set above.

To output degrees/minutes/seconds in a context other than axis tics, such as placing labels on a map, you can use the relative time format specifiers %tH %tM %tS for strptime. See **time_specifiers (p. 151)**, **strptime (p. 39)**.

Xtics logscale

If the **logscale** attribute is set for a tic series along a log-scaled axis, the tic interval is interpreted as a multiplicative factor rather than a constant. For example:

```
# generate a series of tics at y=20 y=200 y=2000 y=20000
set log y
set ytics 20, 10, 50000 logscale
```

Note that no tic is placed at y=50000 because it is not in the series 2*10^x. If the logscale property is disabled, the tic increment will be treated as an additive constant even for a log-scaled axis. For example:

```
# generate a series of tics at y=20 y=40 y=60 ... y=200
set log y
set yrange [20:200]
set ytics 20 nologscale
```

The **logscale** attribute is set automatically by the **set log** command, so normally you do not need this keyword unless you want to force a constant tic interval as in the second example above.

Xtics rangelimited

This option limits both the auto-generated axis tic labels and the corresponding plot border to the range of values actually present in the data that has been plotted. Note that this is independent of the current range limits for the plot. For example, suppose that the data in "file.dat" all lies in the range 2 < y < 4. Then the following commands will create a plot for which the left-hand plot border (y axis) is drawn for only this portion of the total y range, and only the axis tics in this region are generated. I.e., the plot will be scaled to the full range on y, but there will be a gap between 0 and 2 on the left border and another gap between 4 and 10. This style is sometimes referred to as a **range-frame** graph.

```
set border 3
set yrange [0:10]
set ytics nomirror rangelimited
plot "file.dat"
```

Xyplane

The **set xyplane** 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 xyplane at <zvalue>
set xyplane relative <frac>
set ticslevel <frac>        # equivalent to set xyplane relative
show xyplane
```

The form **set xyplane relative <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.

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 **set view (p. 211)**, and **set zeroaxis (p. 224)**.

Xzeroaxis

The **set xzeroaxis** command draws a line at y = 0. For details, please see **set zeroaxis (p. 224)**.

Y2data

The **set y2data** command sets y2 (right-hand) axis data to timeseries (dates/times). Please see **set xdata (p. 213)**.

Y2dtics

The **set y2dtics** command changes tics on the y2 (right-hand) axis to days of the week. Please see **set xdtics (p. 214)** for details.

Y2label

The **set y2label** command sets the label for the y2 (right-hand) axis. Please see **set xlabel (p. 214)**.

Y2mtics

The **set y2mtics** command changes tics on the y2 (right-hand) axis to months of the year. Please see **set xmtics (p. 215)** for details.

Y2range

The **set y2range** command sets the vertical range that will be displayed on the y2 (right) axis. See **set xrange (p. 215)** for the full set of command options. See also **set link (p. 165)**.

Y2tics

The **set y2tics** command controls major (labeled) tics on the y2 (right-hand) axis. Please see **set xtics (p. 217)** for details.

Y2zeroaxis

The **set y2zeroaxis** command draws a line at the origin of the y2 (right-hand) axis (x2 = 0). For details, please see **set zeroaxis (p. 224)**.

Ydata

The **set ydata** commands sets y-axis data to timeseries (dates/times). Please see **set xdata (p. 213)**.

Ydtics

The **set ydtics** command changes tics on the y axis to days of the week. Please see **set xdtics (p. 214)** for details.

Ylabel

This command sets the label for the y axis. Please see **set xlabel (p. 214)**.

Ymtics

The **set ymtics** command changes tics on the y axis to months of the year. Please see **set xmtics (p. 215)** for details.

Yrange

The **set yrange** command sets the vertical range that will be displayed on the y axis. Please see **set xrange (p. 215)** for details.

Ytics

The **set ytics** command controls major (labeled) tics on the y axis. Please see **set xtics (p. 217)** for details.

Yzeroaxis

The **set yzeroaxis** command draws a line at x = 0. For details, please see **set zeroaxis (p. 224)**.

Zdata

The **set zdata** command sets zaxis data to timeseries (dates/times). Please see **set xdata (p. 213)**.

Zdtics

The **set zdtics** command changes tics on the z axis to days of the week. Please see **set xdtics (p. 214)** for details.

Zzeroaxis

The **set zzeroaxis** 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 **set zeroaxis (p. 224)** and **set xyplane (p. 221)**.

Cbdata

Set color box axis data to timeseries (dates/times). Please see **set xdata (p. 213)**.

Cbdtics

The **set cbdtics** command changes tics on the color box axis to days of the week. Please see **set xdtics (p. 214)** for details.

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.

Zeroaxis

The x axis may be drawn by **set xzeroaxis** and removed by **unset xzeroaxis**. Similar commands behave similarly for the y, x2, y2, and z axes. **set zeroaxis ...** (no prefix) acts on the x, y, and z axes jointly.

Syntax:

```
set {x|x2|y|y2|z}zeroaxis { {linestyle | ls <line_style>}
                           | {linetype | lt <line_type>}
                             {linewidth | lw <line_width>}
                             {linecolor | lc <colorspec>}
                             {dashtype | dt <dashtype>} }
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>, width <line_width>, color <colorspec>, and dash type <dashtype> (if supported by the terminal driver currently in use), or a user-defined style <line_style> (see **set style line (p. 200)**).

If no linetype is specified, any zero axes selected will be drawn using the axis linetype (linetype 0).

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
```

Zlabel

This command sets the label for the z axis. Please see **set xlabel (p. 214)**.

Zmtics

The **set zmtics** command changes tics on the z axis to months of the year. Please see **set xmtics (p. 215)** for details.

Zrange

The **set zrange** 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 **set xrange (p. 215)** for details.

Ztics

The **set ztics** command controls major (labeled) tics on the z axis. Please see **set xtics (p. 217)** for details.

Cblabel

This command sets the label for the color box axis. Please see **set xlabel (p. 214)**.

Cbmtics

The **set cbmtics** command changes tics on the color box axis to months of the year. Please see **set xmtics (p. 215)** for details.

Cbrange

The **set cbrange** command sets the range of values which are colored using the current **palette** by styles **with pm3d**, **with image** and **with palette**. Values outside of the color range use color of the nearest extreme.

If the cb-axis is autoscaled in **splot**, then the colorbox range is taken from **zrange**. Points drawn in **splot ... pm3d|palette** can be filtered by using different **zrange** and **cbrange**.

Please see **set xrange (p. 215)** for details on **set cbrange (p. 225)** syntax. See also **set palette (p. 180)** and **set colorbox (p. 139)**.

Cbtics

The **set cbtics** command controls major (labeled) tics on the color box axis. Please see **set xtics (p. 217)** for details.

Shell

The **shell** command spawns an interactive shell. To return to **gnuplot**, type **logout** if using VMS, **exit** or the END-OF-FILE character if using Unix, or **exit** if using MS-DOS or OS/2.

The **shell** command ignores anything else on the gnuplot command line. If instead you want to pass a command string to a shell for immediate execution, use the **system** function or the shortcut **!**. See **system (p. 232)**.

Examples:

```
shell
system "print previous_plot.ps"
! print previous_plot.ps
current_time = system("date")
```

Splot

splot is the command for drawing 3D plots (well, actually projections on a 2D surface, but you knew that). It is the 3D equivalent of the **plot** command. **splot** provides only a single x, y, and z axis; there is no equivalent to the x2 and y2 secondary axes provided by **plot**.

See the **plot (p. 101)** command for many options available in both 2D and 3D plots.

Syntax:

```
splot {<ranges>}
      {<iteration>}
      <function> | {{<file name> | <datablock name>} {datafile-modifiers}}
```

```
                    | <voxelgridname>
                    | keyentry
          {<title-spec>} {with <style>}
          {, {definitions{,}} <function> ...}
```

The **splot** command operates on a data generated by a function, read from a data file, or stored previously in a named data block. Data file names are usually provided as a quoted string. The function can be a mathematical expression, or a triple of mathematical expressions in parametric mode.

A new feature in version 5.4 is that **splot** can operate on voxel data. See **voxel-grids (p. 230)**, **set vgrid (p. 210)**, **vxrange (p. 212)**. At present voxel grids can be be plotted using styles **with dots**, **with points**, or **with isosurface**. Voxel grid values can also be referenced in the **using** specifiers of other plot styles, for example to assign colors.

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 **set xyplane**. The orientation of a **splot** projection is controlled by **set view**. See **set view (p. 211)** and **set xyplane (p. 221)** for more information.

The syntax for setting ranges on the **splot** command is the same as for **plot**. In non-parametric mode, ranges must be given in the order

```
splot [<xrange>][<yrange>][<zrange>] ...
```

In parametric mode, the order is

```
splot [<urange>][<vrange>][<xrange>][<yrange>][<zrange>] ...
```

The **title** option is the same as in **plot**. The operation of **with** is also the same as in **plot** except that not all 2D plotting styles are available.

The **datafile** options have more differences.

As an alternative to surfaces drawn using parametric or function mode, the pseudo-file '++' can be used to generate samples on a grid in the xy plane.

See also **show plot (p. 187)**, **set view map (p. 211)**, and **sampling (p. 121)**.

Data-file

Splot, like **plot**, can display from a file.

Syntax:

```
splot '<file_name>' {binary <binary list>}
                    {{nonuniform} matrix}
                    {index <index list>}
                    {every <every list>}
                    {using <using list>}
```

The special filenames `""` and `"-"` are permitted, as in **plot**. See **special-filenames (p. 113)**.

In brief, **binary** and **matrix** indicate that the data are in a special form, **index** selects which data sets in a multi-data-set file are to be plotted, **every** specifies which datalines (subsets) within a single data set are to be plotted, and **using** determines how the columns within a single record are to be interpreted.

The options **index** and **every** behave the same way as with **plot**; **using** does so also, except that the **using** list must provide three entries instead of two.

The **plot** option **smooth** is not available for **splot**, but **cntrparam** and **dgrid3d** 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 block number will be used for y, and the index of the data point in the block 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 blocks of data in a **splot** datafile; **splot** treats blocks as the equivalent of function y-isolines. No line will join points separated by a blank record. If all blocks contain the same number of points, **gnuplot** will draw cross-isolines between points in the blocks, connecting corresponding points. This is termed "grid data", and is required for drawing a surface, for contouring (**set contour**) and hidden-line removal (**set hidden3d**). See also **splot grid_data (p. 229)**.

It is no longer necessary to specify **parametric** mode for three-column **splot**s.

Matrix

Gnuplot can interpret matrix data input in two different ways.

The first of these assumes a uniform grid of x and y coordinates and assigns each value in the input matrix to one element M[i,j] of this uniform grid. The assigned x coordinates are the integers [0:NCOLS-1]. The assigned y coordinates are the integers [0:NROWS-1]. This is the default for text data input, but not for binary input. See **matrix uniform (p. 227)** for examples and additional keywords.

The second interpretation assumes a non-uniform grid with explicit x and y coordinates. The first row of input data contains the y coordinates; the first column of input data contains the x coordinates. For binary input data, the first element of the first row must contain the number of columns. This is the default for **binary matrix** input, but requires an additional keyword **nonuniform** for text input data. See **matrix nonuniform (p. 227)** for examples.

Uniform Example commands for plotting uniform matrix data:

```
splot 'file' matrix using 1:2:3          # text input
splot 'file' binary general using 1:2:3  # binary input
```

In a uniform grid matrix 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.

For text input, if the first row contains column labels rather than data, use the additional keyword **columnheaders**. Similarly if the first field in each row contains a label rather than data, use the additional keyword **rowheaders**. Here is an example that uses both:

```
$DATA << EOD
xxx A   B   C   D
aa  z11 z12 z13 z14
bb  z21 z22 z23 z24
cc  z31 z32 z33 z34
EOD
plot $DATA matrix columnheaders rowheaders with image
```

For text input, a blank line or comment line ends the matrix, and starts a new data block. You can select among the data blocks in a file by the **index** option to the **splot** command, as usual. The columnheaders option, if present, is applied only to the first data block.

Nonuniform The first row of input data contains the y coordinates. The first column of input data contains the x coordinates. For binary input data, the first field of the first row must contain the number of columns. (This number is ignored for text input).

Example commands for plotting non-uniform matrix data:

```
splot 'file' nonuniform matrix using 1:2:3  # text input
splot 'file' binary matrix using 1:2:3      # binary input
```

Thus the data organization for non-uniform matrix input is

```
<N+1>  <x0>   <x1>   <x2>  ...  <xN>
 <y0> <z0,0> <z0,1> <z0,2> ... <z0,N>
 <y1> <z1,0> <z1,1> <z1,2> ... <z1,N>
  :      :      :      :   ...    :
```

which is then 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.

Every The **every** keyword has special meaning when used with matrix data. Rather than applying to blocks of single points, it applies to rows and column values. Note that matrix rows and columns are indexed starting from 0, so the row with index N is the (N+1)th row. Syntax:

```
plot 'file' every {<column_incr>}
                    {:{<row_incr>}
                      {:{<start_column>}
                        {:{<start_row>}
                          {:{<end_column>}
                            {:<end_row>}}}}}
```

Examples:

```
plot 'file' matrix every :::N::N   # plot all values in row with index N
plot 'file' matrix every ::3::7    # plot columns 3 to 7 for all rows
plot 'file' matrix every ::3:0:7:4 # submatrix bounded by [3,0] and [7,4]
```

Examples 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**.

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 **every** option you can specify explicit rows and columns.

Example – rescale axes of a matrix in a text file:

```
splot ‘a.dat‘ matrix using (1+$1):(1+$2*10):3
```

Example – plot the 3rd row of a matrix in a text file:

```
plot 'a.dat' matrix using 1:3 every 1:999:1:2
```

(rows are enumerated from 0, thus 2 instead of 3).

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 (p. 103)** for more details.)

Example datafile

A simple example of plotting a 3D 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 (blocks) 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 block, 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 block. 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.

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 **set isosamples (p. 156)**, or reading a datafile, see **splot datafile (p. 226)**. 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 **set isosamples (p. 156)**.

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 block, then "isolines" will connect the points in a block, and "cross-isolines" will connect the corresponding points in each block 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 **set dgrid3d**.

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
```

Splot surfaces

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 **set isosamples (p. 156)**. The surface may be approximated by connecting the points with straight line segments, see **set surface (p. 204)**, 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 **set view**.

Additionally, for points in a grid format, **splot** can interpolate points having a common amplitude (see **set contour (p. 140)**) and can then connect those new points to display contour lines, either directly with straight-line segments or smoothed lines (see **set cntrparam (p. 137)**). Functions are already evaluated in a grid format, determined by **set isosamples** and **set samples**, while file data must either be in a grid format, as described in **data-file**, or be used to generate a grid (see **set dgrid3d (p. 144)**).

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.

Voxel-grid

Syntax:

```
splot $voxelgridname with {dots|points} {above <threshold>} ...
splot $voxelgridname with isosurface {level <threshold>} ...
```

Voxel data can be plotted with dots or points marking individual voxels whose value is above the specified threshold value (default threshold = 0). Color/pointtype/linewidth properties can be appended as usual.

At many view angles the voxel grid points will occlude each other or create Moiré artifacts on the display. These effects can be avoided by introducing jitter so that the displayed dot or point is displaced randomly from the true voxel grid coordinate. See **set jitter (p. 156)**.

Dense voxel grids can be down-sampled by using the **pointinterval** property (**pi** for short) to reduce the number of points drawn.

```
splot $vgrid with points pointtype 6 pointinterval 2
```

with isosurface will create a tessellated surface in 3D enclosing all voxels with value greater than the requested threshold. The surface placement is adjusted by linear interpolation to pass through the threshold value itself.

See **set vgrid (p. 210)**, **vfill (p. 235)**. See demos **vplot.dem**, **isosurface.dem**.

Stats (Statistical Summary)

Syntax:

```
stats {<ranges>} 'filename' {matrix | using N{:M}} {name 'prefix'} {{no}output}
```

This command prepares a statistical summary of the data in one or two columns of a file. The using specifier is interpreted in the same way as for plot commands. See **plot (p. 101)** for details on the **index (p. 110)**, **every (p. 108)**, and **using (p. 115)** directives. Data points are filtered against both xrange and yrange before analysis. See **set xrange (p. 215)**. The summary is printed to the screen by default. Output can be redirected to a file by prior use of the command **set print**, or suppressed altogether using the **nooutput** option.

In addition to printed output, the program stores the individual statistics into three sets of variables. The first set of variables reports how the data is laid out in the file: The array of column headers is generated only if option **set datafile columnheaders** is in effect.

`STATS_records`	N	total number N of in-range data records
`STATS_outofrange`		number of records filtered out by range limits
`STATS_invalid`		number of invalid/incomplete/missing records
`STATS_blank`		number of blank lines in the file
`STATS_blocks`		number of indexable blocks of data in the file
`STATS_columns`		number of data columns in the first row of data
`STATS_column_header`		array of strings holding column headers found

The second set reports properties of the in-range data from a single column. This column is treated as y. If the y axis is autoscaled then no range limits are applied. Otherwise only values in the range [ymin:ymax] are considered.

If two columns are analysed jointly by a single **stats** command, the suffix "_x" or "_y" is appended to each variable name. I.e. STATS_min_x is the minimum value found in the first column, while STATS_min_y is the minimum value found in the second column. In this case points are filtered by testing against both xrange and yrange.

`STATS_min`		$\min(y)$	minimum value of in-range data points
`STATS_max`		$\max(y)$	maximum value of in-range data points
`STATS_index_min`		$i \mid y_i = \min(y)$	index i for which data[i] == STATS_min
`STATS_index_max`		$i \mid y_i = \max(y)$	index i for which data[i] == STATS_max
`STATS_mean`	$\bar{y} =$	$\frac{1}{N}\sum y$	mean value of the in-range data points
`STATS_stddev`	$\sigma_y =$	$\sqrt{\frac{1}{N}\sum (y-\bar{y})^2}$	population standard deviation of the in-range data
`STATS_ssd`	$s_y =$	$\sqrt{\frac{1}{N-1}\sum (y-\bar{y})^2}$	sample standard deviation of the in-range data
`STATS_lo_quartile`			value of the lower (1st) quartile boundary
`STATS_median`			median value
`STATS_up_quartile`			value of the upper (3rd) quartile boundary
`STATS_sum`		$\sum y$	sum
`STATS_sumsq`		$\sum y^2$	sum of squares
`STATS_skewness`		$\frac{1}{N\sigma^3}\sum (y-\bar{y})^3$	skewness of the in-range data points
`STATS_kurtosis`		$\frac{1}{N\sigma^4}\sum (y-\bar{y})^4$	kurtosis of the in-range data points
`STATS_adev`		$\frac{1}{N}\sum \|y-\bar{y}\|$	mean absolute deviation of the in-range data
`STATS_mean_err`		$\sigma_y/\sqrt{N}$	standard error of the mean value
`STATS_stddev_err`		$\sigma_y/\sqrt{2N}$	standard error of the standard deviation
`STATS_skewness_err`		$\sqrt{6/N}$	standard error of the skewness
`STATS_kurtosis_err`		$\sqrt{24/N}$	standard error of the kurtosis

The third set of variables is only relevant to analysis of two data columns.

`STATS_correlation`	sample correlation coefficient between x and y values
`STATS_slope`	A corresponding to a linear fit y = Ax + B
`STATS_slope_err`	uncertainty of A
`STATS_intercept`	B corresponding to a linear fit y = Ax + B
`STATS_intercept_err`	uncertainty of B
`STATS_sumxy`	sum of x*y
`STATS_pos_min_y`	x coordinate of a point with minimum y value
`STATS_pos_max_y`	x coordinate of a point with maximum y value

Keyword **matrix** indicates that the input consists of a matrix (see **matrix (p. 227)**); the usual statistics are generated by considering all matrix elements. The matrix dimensions are saved in variables STATS_size_x and STATS_size_y.

`STATS_size_x`	number of matrix columns
`STATS_size_y`	number of matrix rows

The index reported in STATS_index_xxx corresponds to the value of pseudo-column 0 ($0) in plot commands. I.e. the first point has index 0, the last point has index N-1.

Data values are sorted to find the median and quartile boundaries. If the total number of points N is odd, then the median value is taken as the value of data point (N+1)/2. If N is even, then the median is reported as the mean value of points N/2 and (N+2)/2. Equivalent treatment is used for the quartile boundaries.

For an example of using the **stats** command to annotate a subsequent plot, see `stats.dem.`

The **stats** command in this version of gnuplot can handle log-scaled data, but not the content of time/date fields (**set xdata time** or **set ydata time**). This restriction may be relaxed in a future version.

Name

It may be convenient to track the statistics from more than one file or data column in parallel. The **name** option causes the default prefix "STATS" to be replaced by a user-specified string. For example, the mean value of column 2 data from two different files could be compared by

```
stats "file1.dat" using 2 name "A"
stats "file2.dat" using 2 name "B"
if (A_mean < B_mean) {...}
```

Instead of providing a string constant as the name, the keyword **columnheader** or function **columnheader(N)** can be used to generate the name from whatever string is found in that column in the first row of the data file:

```
do for [COL=5:8] { stats 'datafile' using COL name columnheader }
```

System

Syntax:

```
system "command string"
! command string
output = system("command string")
show variable GPVAL_SYSTEM
```

system "command" executes "command" in a subprocess by invoking the operating system's default shell. If called as a function, **system("command")** returns the character stream from the subprocess's stdout as a string. One trailing newline is stripped from the resulting string if present. See also **backquotes (p. 57)**.

The exit status of the subprocess is reported in variables GPVAL_SYSTEM_ERRNO and GPVAL_SYSTEM_ERRMSG. Note that if the command string invokes more than one programs, the subprocess may return "Success" even if one of the programs produced an error. E.g. file = system("ls -1 *.plt | tail -1") will return "Success" even if there are no *.plt files because **tail** succeeds even if **ls** does not.

The system command can be used to import external functions into gnuplot as shown below, however this will force creation of a separate subprocess every time the function is invoked. For functions that will be invoked many times it would be better to import a directly callable subroutine from a shared library. See **import (p. 99)** and **plugin.dem**.

```
f(x) = real(system(sprintf("somecommand %f", x)))
```

Test

This command graphically tests or presents terminal and palette capabilities.

Syntax:

```
test {terminal | palette}
```

test or **test terminal** creates a display of line and point styles and other useful things supported by the **terminal** you are currently using.

test palette plots profiles of R(z),G(z),B(z), where 0<=z<=1. These are the RGB components of the current color **palette**. It also plots the apparent net intensity as calculated using NTSC coefficients to map RGB onto a grayscale. The profile values are also loaded into a datablock named $PALETTE.

Toggle

Syntax:

```
toggle {<plotno> | "plottitle" | all}
```

This command has the same effect as left-clicking on the key entry for a plot currently displayed by an interactive terminal (qt, wxt, x11). If the plot is showing, it is toggled off; if it is currently hidden, it is toggled on. **toggle all** acts on all active plots, equivalent to the hotkey "i". **toggle "title"** requires an exact match to the plot title. **toggle "ti*"** acts on the first plot whose title matches the characters before the final '*'. If the current terminal is not interactive, the toggle command has no effect.

Undefine

Clear one or more previously defined user variables. This is useful in order to reset the state of a script containing an initialization test.

A variable name can contain the wildcard character * as last character. If the wildcard character is found, all variables with names that begin with the prefix preceding the wildcard will be removed. This is useful to remove several variables sharing a common prefix. Note that the wildcard character is only allowed at the end of the variable name! Specifying the wildcard character as sole argument to **undefine** has no effect.

Example:

```
undefine foo foo1 foo2
if (!exists("foo")) load "initialize.gp"

bar = 1; bar1 = 2; bar2 = 3
undefine bar*                  # removes all three variables
```

Unset

Options set using the **set** command may be returned to their default state by the corresponding **unset** command. The **unset** command may contain an optional iteration clause. See **plot for (p. 122)**.

Examples:

```
set xtics mirror rotate by -45 0,10,100
...
unset xtics
# Unset labels numbered between 100 and 200
unset for [i=100:200] label i
```

Linetype

Syntax:

```
unset linetype N
```

Remove all characteristics previously associated with a single linetype. Subsequent use of this linetype will use whatever characteristics and color that is native to the current terminal type (i.e. the default linetypes properties available in gnuplot versions prior to 4.6).

Monochrome

Switches the active set of linetypes from monochrome to color. Equivalent to **set color**.

Output

Because some terminal types allow multiple plots to be written into a single output file, the output file is not automatically closed after plotting. In order to print or otherwise use the file safely, it should first be closed explicitly by using **unset output** or by using **set output** to close the previous file and then open a new one.

Terminal

The default terminal that is active at the time of program entry depends on the system platform, gnuplot build options, and the environmental variable GNUTERM. Whatever this default may be, gnuplot saves it to internal variable GNUTERM. The **unset terminal** command restores the initial terminal type. It is equivalent to **set terminal GNUTERM**. However if the string in GNUTERM contains terminal options in addition to the bare terminal name, you may want to instead use **set terminal @GNUTERM**.

Update

Note: This command is DEPRECATED. Use **save fit** instead.

Vclear

Syntax:

```
vclear {$gridname}
```

Resets the value of all voxels in an existing grid to zero. If no grid name is given, clears the currently active grid.

Vfill

Syntax:

```
vfill FILE using x:y:z:radius:(<expression>)
```

The **vfill** command acts analogously to a **plot** command except that instead of creating a plot it modifies voxels in the currently active voxel grid. For each point read from the input file, the voxel containing that point and also all other voxels within a sphere of given radius centered about (x,y,z) are incremented as follows:

- user variable VoxelDistance is set to the distance from (x,y,z) to that voxel's grid coordinates (vx,vy,vz).
- The expression provided in the 5th **using** specifier is evaluated. This expression can use the new value of VoxelDistance.
- voxel(vx,vy,vz) += result of evaluating <expression>

Example:

```
vfill "file.dat" using 1:2:3:(3.0):(1.0)
```

This command adds 1 to the value of every voxel within a sphere of radius 3.0 around each point in file.dat.

Example:

```
vfill "file.dat" using 1:2:3:4:(VoxelDistance < 1 ? 1 : 1/VoxelDistance)
```

This command modifies all voxels in a sphere whose radius is determined for each point by the content of column 4. The increment added to a voxel decreases with its distance from the data point.

Note that **vfill** always increments existing values in the current voxel grid. To reset them to zero, use **vclear**.

While

Syntax:

```
while (<expr>) {
    <commands>
}
```

Execute a block of commands repeatedly so long as <expr> evaluates to a non-zero value. This command cannot be mixed with old-style (un-bracketed) if/else statements. See also **do (p. 89)**, **continue (p. 88)**, **break (p. 86)**.

Part IV

Terminal types

Complete list of terminals

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 (p. 205)**.

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.

Terminals marked **legacy** are not built by default in recent gnuplot versions and may not actually work.

Aifm

NOTE: Legacy terminal, originally written for Adobe Illustrator 3.0+. Since Adobe Illustrator understands PostScript level 1 commands directly, you should use **set terminal post level1** instead.

Syntax:

```
set terminal aifm {color|monochrome} {"<fontname>"} {<fontsize>}
```

Aqua

This terminal relies on AquaTerm.app for display on Mac OS X.

Syntax:

```
set terminal aqua {<n>} {title "<wintitle>"} {size <x> <y>}
                  {font "<fontname>{,<fontsize>}"}
                  {linewidth <lw>}"}
                  {{no}enhanced} {solid|dashed} {dl <dashlength>}}
```

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 <fontname> to specify the font (default is "Times-Roman"), and <fontsize> to specify the font size (default is 14.0 pt).

The aqua terminal supports enhanced text mode (see **enhanced (p. 34)**), except for overprint. Font support is limited to the fonts available on the system. Character encoding can be selected by **set encoding** and currently supports iso_latin_1, iso_latin_2, cp1250, and UTF8 (default).

Lines can be drawn either solid or dashed, (default is solid) and the dash spacing can be modified by <dashlength> which is a multiplier > 0.

Be

The **be** terminal type is present if gnuplot is built for the **beos** operating system and for use with X servers. It is selected at program 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 **reset**, 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 rendering on grayscale or color displays. (Grayscale displays receive monochrome rendering 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

Caca

[EXPERIMENTAL] The **caca** terminal is a mostly-for-fun output mode that uses **libcaca** to plot using ascii characters. In contrast to the **dumb** terminal it includes support for color, box fill, images, rotated text, filled polygons, and mouse interaction.

Syntax:

```
set terminal caca {{driver | format} {default | <driver> | list}}
                  {color | monochrome}
                  {{no}inverted}
                  {enhanced | noenhanced}
                  {background <rgb color>}
                  {title "<plot window title>"}
                  {size <width>,<height>}
                  {charset ascii|blocks|unicode}
```

The **driver** option selects the **libcaca** display driver or export **format**. Use **default** is to let **libcaca** choose the platform default display driver. The default driver can be changed by setting the environment variable CACA_DRIVER before starting **gnuplot**. Use **set term caca driver list** to print a list of supported output modes.

The **color** and **monochrome** options select colored or mono output. Note that this also changes line symbols. Use the **inverted** option if you prefer a black background over the default white. This also changes the color of black default linetypes to white.

Enhanced text support can be activated using the **enhanced** option, see **enhanced text** **(p. 34)**.

The title of the output window can be changed with the **title** option, if supported by the **libcaca** driver.

The **size** option selects the size of the canvas in characters. The default is 80 by 25. If supported by the backend, the canvas size will be automatically adjusted to the current window/terminal size. The default size of the "x11" and "gl" window can be controlled via the CACA_GEOMETRY environment variable. The geometry of the window of the "win32" driver can be controlled and permanently changed via the app menu.

The **charset** option selects the character set used for lines, points, filling of polygons and boxes and dithering of images. Note that some backend/terminal/font combinations might not support some characters of the **blocks** or **unicode** character set. On Windows it is recommend to use a non-raster font such as "Lucida Console" or "Consolas".

The caca terminal supports mouse interaction. Please beware that some backends of **libcaca** (e.g. slang, ncurses) only update the mouse position on mouse clicks. Modifier keys (ctrl, alt, shift) are not supported by **libcaca** and are thus unavailable.

The default **encoding** of the **caca** terminal is utf8. It also supports the cp437 **encoding**.

The number of colors supported by **libcaca** backends differs. Most backends support 16 foreground and 16 background colors only, whereas e.g. the "x11" backend supports truecolor.

Depending on the terminal and **libcaca** backend, only 8 different background colors might be supported. Bright colors (with the most most significant bit of the background color set) are then interpreted as indicator for blinking text. Try using **background rgb "gray"** in that case.

See also the libcaca web site at http://caca.zoy.org/wiki/libcaca

and libcaca environment variables http://caca.zoy.org/doxygen/libcaca/libcaca-env.html

Caca limitations and bugs

The **caca** terminal has known bugs and limitations:

Unicode support depends on the driver and the terminal. The "x11" backend supports unicode since libcaca version 0.99.beta17. Due to a bug in **libcaca** <0.99.beta20, the "slang" driver does not support unicode. Note that **libcaca** <0.99.beta19 contains a bug which results in an endless loop if you supply illegal 8bit sequences.

Bright background colors may cause blinking.

Modifier keys are not supported for mousing, see **term caca (p. 239)**.

Rotated enhanced text, and transparency are not supported. The **size** option is not considered for on-screen display.

In order to correctly draw the key box, use

```
set key width 1 height 1
```

Alignment of enhanced text is wrong if it contains utf8 characters. Resizing of Windows console window does not work correctly due to a bug in libcaca. Closing the terminal window by clicking the "X" on the title line will terminate wgnuplot. Press "q" to close the window.

Cairolatex

The **cairolatex** terminal device generates encapsulated PostScript (*.eps), PDF, or PNG output using the cairo and pango support libraries and uses LaTeX for text output using the same routines as the **epslatex** terminal.

Syntax:

```
set terminal cairolatex
               {eps | pdf | png}
               {standalone | input}
               {blacktext | colortext | colourtext}
               {header <header> | noheader}
               {mono|color}
               {{no}transparent} {{no}crop} {background <rgbcolor>}
               {font <font>} {fontscale <scale>}
               {linewidth <lw>} {rounded|butt|square} {dashlength <dl>}
               {size <XX>{unit},<YY>{unit}}
               {resolution <dpi>}
```

The cairolatex terminal prints a plot like **terminal epscairo** or **terminal pdfcairo** but transfers the texts to LaTeX instead of including them in the graph. For reference of options not explained here see **pdfcairo (p. 271)**.

eps, **pdf**, or **png** select the type of graphics output. Use **eps** with latex/dvips and **pdf** for pdflatex. If your plot has a huge number of points use **png** to keep the filesize down. When using the **png** option, the terminal accepts an extra option **resolution** to control the pixel density of the resulting PNG. The argument of **resolution** is an integer with the implied unit of DPI.

blacktext forces all text to be written in black even in color mode;

The **cairolatex** driver offers a special way of controlling text positioning: (a) 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. (b) If the text string begins with '[', you need to continue it with: a position specification (up to two out of t,b,l,r,c), ']{', 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. See also the documentation for the **pslatex (p. 280)** terminal driver. To create multiline labels, use \shortstack, for example

```
set ylabel '[r]{\shortstack{first line \\ second line}}'
```

The **back** option of **set label** commands is handled slightly different than in other terminals. Labels using 'back' are printed behind all other elements of the plot while labels using 'front' are printed above everything else.

The driver produces two different files, one for the eps, pdf, or png part of the figure and one for the LaTeX part. The name of the LaTeX file is taken from the **set output** command. The name of the eps/pdf/png file is derived by replacing the file extension (normally '.tex') with '.eps'/'.pdf'/'.png' instead. There is no LaTeX output if no output file is given! Remember to close the **output file** before next plot unless in **multiplot** mode.

In your LaTeX documents use '\input{filename}' to include the figure. The '.eps'/'.pdf'/'.png' file is included by the command \includegraphics{...}, so you must also include \usepackage{graphicx} in the LaTeX preamble. If you want to use coloured text (option **colourtext**) you also have to include \usepackage{color} in the LaTeX preamble.

The behaviour concerning font selection depends on the header mode. In all cases, the given font size is used for the calculation of proper spacing. When not using the **standalone** mode the actual LaTeX font and font size at the point of inclusion is taken, so use LaTeX commands for changing fonts. If you use e.g. 12pt as font size for your LaTeX document, use '", 12"' as options. The font name is ignored. If using **standalone** the given font and font size are used, see below for a detailed description.

If text is printed coloured is controlled by the TeX booleans \ifGPcolor and \ifGPblacktext. Only if \ifGPcolor is true and \ifGPblacktext is false, text is printed coloured. You may either change them in the generated TeX file or provide them globally in your TeX file, for example by using

```
\newif\ifGPblacktext
\GPblacktexttrue
```

in the preamble of your document. The local assignment is only done if no global value is given.

When using the cairolatex terminal give the name of the TeX file in the **set output** command including the file extension (normally ".tex"). The graph filename is generated by replacing the extension.

If using the **standalone** mode a complete LaTeX header is added to the LaTeX file; and "-inc" is added to the filename of the gaph file. The **standalone** mode generates a TeX file that produces output with the correct size when using dvips, pdfTeX, or VTeX. The default, **input**, generates a file that has to be included into a LaTeX document using the \input command.

If a font other than "" or "default" is given it is interpreted as LaTeX font name. It contains up to three parts, separated by a comma: 'fontname,fontseries,fontshape'. If the default fontshape or fontseries are requested, they can be omitted. Thus, the real syntax for the fontname is '{fontname}{,fontseries}{,fontshape}'. The naming convention for all parts is given by the LaTeX font scheme. The fontname is 3 to 4 characters long and is built as follows: One character for the font vendor, two characters for the name of the font, and optionally one additional character for special fonts, e.g., 'j' for fonts with old-style numerals or 'x' for expert fonts. The names of many fonts is described in `http://www.tug.org/fontname/fontname.pdf`

For example, 'cmr' stands for Computer Modern Roman, 'ptm' for Times-Roman, and 'phv' for Helvetica. The font series denotes the thickness of the glyphs, in most cases 'm' for normal ("medium") and 'bx' or 'b' for bold fonts. The font shape is 'n' for upright, 'it' for italics, 'sl' for slanted, or 'sc' for small caps, in general. Some fonts may provide different font series or shapes.

Examples:

Use Times-Roman boldface (with the same shape as in the surrounding text):

```
set terminal cairolatex font 'ptm,bx'
```

Use Helvetica, boldface, italics:

```
set terminal cairolatex font 'phv,bx,it'
```

Continue to use the surrounding font in slanted shape:

```
set terminal cairolatex font ',,sl'
```

Use small capitals:

```
set terminal cairolatex font ',,sc'
```

By this method, only text fonts are changed. If you also want to change the math fonts you have to use the "gnuplot.cfg" file or the **header** option, described below.

In **standalone** mode, the font size is taken from the given font size in the **set terminal** command. To be able to use a specified font size, a file "size<size>.clo" has to reside in the LaTeX search path. By default, 10pt, 11pt, and 12pt are supported. If the package "extsizes" is installed, 8pt, 9pt, 14pt, 17pt, and 20pt are added.

The **header** option takes a string as argument. This string is written into the generated LaTeX file. If using the **standalone** mode, it is written into the preamble, directly before the \begin{document} command. In the **input** mode, it is placed directly after the \begingroup command to ensure that all settings are local to the plot.

Examples:

Use T1 fontencoding, change the text and math font to Times-Roman as well as the sans-serif font to Helvetica:

```
set terminal cairolatex standalone header \
"\\usepackage[T1]{fontenc}\n\\usepackage{mathptmx}\n\\usepackage{helvet}"
```

Use a boldface font in the plot, not influencing the text outside the plot:

```
set terminal cairolatex input header "\\bfseries"
```

If the file "gnuplot.cfg" is found by LaTeX it is input in the preamble the LaTeX document, when using **standalone** mode. It can be used for further settings, e.g., changing the document font to Times-Roman, Helvetica, and Courier, including math fonts (handled by "mathptmx.sty"):

```
\usepackage{mathptmx}
\usepackage[scaled=0.92]{helvet}
\usepackage{courier}
```

The file "gnuplot.cfg" is loaded before the header information given by the **header** command. Thus, you can use **header** to overwrite some of settings performed using "gnuplot.cfg"

Canvas

The **canvas** terminal creates a set of javascript commands that draw onto the HTML5 canvas element. Syntax:

```
set terminal canvas {size <xsize>, <ysize>} {background <rgb_color>}
                    {font {<fontname>}{,<fontsize>}} | {fsize <fontsize>}
                    {{no}enhanced} {linewidth <lw>}
                    {rounded | butt | square}
                    {dashlength <dl>}
                    {standalone {mousing} | name '<funcname>'}
                    {jsdir 'URL/for/javascripts'}
                    {title '<some string>'}
```

where <xsize> and <ysize> set the size of the plot area in pixels. The default size in standalone mode is 600 by 400 pixels. The default font size is 10.

NB: Only one font is available, the ascii portion of Hershey simplex Roman provided in the file canvastext.js. You can replace this with the file canvasmath.js, which contains also UTF-8 encoded Hershey simplex Greek and math symbols. For consistency with other terminals, it is also possible to use **font "name,size"**. Currently the font **name** is ignored, but browser support for named fonts is likely to arrive eventually.

The default **standalone** mode creates an html page containing javascript code that renders the plot using the HTML 5 canvas element. The html page links to two required javascript files 'canvastext.js' and 'gnuplot_common.js'. An additional file 'gnuplot_dashedlines.js' is needed to support dashed lines. By default these point to local files, on unix-like systems usually in directory /usr/local/share/gnuplot/<version>/js. See installation notes for other platforms. You can change this by using the **jsdir** option to specify either a different local directory or a general URL. The latter is usually appropriate if the plot is exported for viewing on remote client machines.

All plots produced by the canvas terminal are mouseable. The additional keyword **mousing** causes the **standalone** mode to add a mouse-tracking box underneath the plot. It also adds a link to a javascript file 'gnuplot_mouse.js' and to a stylesheet for the mouse box 'gnuplot_mouse.css' in the same local or URL directory as 'canvastext.js'.

The **name** option creates a file containing only javascript. Both the javascript function it contains and the id of the canvas element that it draws onto are taken from the following string parameter. The commands

```
set term canvas name 'fishplot'
set output 'fishplot.js'
```

will create a file containing a javascript function fishplot() that will draw onto a canvas with id=fishplot. An html page that invokes this javascript function must also load the canvastext.js function as described above. A minimal html file to wrap the fishplot created above might be:

```
<html>
<head>
    <script src="canvastext.js"></script>
    <script src="gnuplot_common.js"></script>
</head>
<body onload="fishplot();">
    <script src="fishplot.js"></script>
    <canvas id="fishplot" width=600 height=400>
        <div id="err_msg">No support for HTML 5 canvas element</div>
    </canvas>
</body>
</html>
```

The individual plots drawn on this canvas will have names fishplot_plot_1, fishplot_plot_2, and so on. These can be referenced by external javascript routines, for example gnuplot.toggle_visibility("fishplot_plot_2").

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".

Syntax:

```
     set terminal cgm {color | monochrome} {solid | dashed} {{no}rotate}
                      {<mode>} {width <plot_width>} {linewidth <line_width>}
                      {font "<fontname>,<fontsize>"}
                      {background <rgb_color>}
 [deprecated]         {<color0> <color1> <color2> ...}
```

solid draws all curves with solid lines, overriding any dashed patterns; <mode> is **landscape**, **portrait**, or **default**; <plot_width> is the assumed width of the plot in points; <line_width> is the line width in points (default 1); <fontname> is the name of a font (see list of fonts below) <fontsize> is the size of the font in points (default 12).

The first six options can be in any order. Selecting **default** sets all options to their default values.

The mechanism of setting line colors in the **set term** command is deprecated. Instead you should set the background using a separate keyword and set the line colors using **set linetype**. The deprecated mechanism accepted colors of the form 'xrrggbb', where x is the literal character 'x' and 'rrggbb' are the red, green and blue components in hex. The first color was used for the background, subsequent colors are assigned to successive line types.

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!
```

Cgm 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):

CGM fonts	
Helvetica	Hershey/Cartographic_Roman
Helvetica Bold	Hershey/Cartographic_Greek
Helvetica Oblique	Hershey/Simplex_Roman
Helvetica Bold Oblique	Hershey/Simplex_Greek
Times Roman	Hershey/Simplex_Script
Times Bold	Hershey/Complex_Roman
Times Italic	Hershey/Complex_Greek
Times Bold Italic	Hershey/Complex_Italic
Courier	Hershey/Complex_Cyrillic
Courier Bold	Hershey/Duplex_Roman
Courier Oblique	Hershey/Triplex_Roman
Courier Bold Oblique	Hershey/Triplex_Italic
Symbol	Hershey/Gothic_German
ZapfDingbats	Hershey/Gothic_English
Script	Hershey/Gothic_Italian
15	Hershey/Symbol_Set_1
	Hershey/Symbol_Set_2
	Hershey/Symbol_Math

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 **set label** command.

Cgm fontsize

Fonts are scaled assuming the page is 6 inches wide. If the **size** 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.

Cgm 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.

Cgm 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.

Cgm 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 line type.

Cgm 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.

Cgm 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
```

Cgm 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. The **nofontlist** (synonym **winword6**) 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.

Context

ConTeXt is a macro package for TeX, highly integrated with Metapost (for drawing figures) and intended for creation of high-quality PDF documents. The terminal outputs Metafun source, which can be edited manually, but you should be able to configure most things from outside.

For an average user of ConTeXt + gnuplot module it's recommended to refer to **Using ConTeXt** rather than reading this page or to read the manual of the gnuplot module for ConTeXt.

The **context** terminal supports the following options:

Syntax:

```
set term context {default}
        {defaultsize | size <scale> | size <xsize>{in|cm}, <ysize>{in|cm}}
        {input | standalone}
        {timestamp | notimestamp}
        {noheader | header "<header>"}
        {color | colour | monochrome}
        {rounded | mitered | beveled} {round | butt | squared}
        {dashed | solid} {dashlength | dl <dl>}
        {linewidth | lw <lw>}
        {fontscale <fontscale>}
        {mppoints | texpoints}
        {inlineimages | externalimages}
        {defaultfont | font "{<fontname>}{,<fontsize>}"}
```

In non-standalone (**input**) graphic only parameters **size** to select graphic size, **fontscale** to scale all the labels for a factor <fontscale> and font size, make sense, the rest is silently ignored and should be configured in the .tex file which inputs the graphic. It's highly recommended to set the proper fontsize if document font differs from 12pt, so that gnuplot will know how much space to reserve for labels.

default resets all the options to their default values.

defaultsize sets the plot size to 5in,3in. **size** <scale> sets the plot size to <scale> times <default value>. If two arguments are given (separated with ','), the first one sets the horizontal size and the second one the vertical size. Size may be given without units (in which case it means relative to the default value), with inches ('in') or centimeters ('cm').

input (default) creates a graphic that can be included into another ConTeXt document. **standalone** adds some lines, so that the document might be compiled as-is. You might also want to add **header** in that case.

Use **header** for any additional settings/definitions/macros that you might want to include in a standalone graphic. **noheader** is the default.

notimestamp prevents printing creation time in comments (if version control is used, one may prefer not to commit new version when only date changes).

color to make color plots is the default, but **monochrome** doesn't do anything special yet. If you have any good ideas how the behaviour should differ to suit the monochrome printers better, your suggestions are welcome.

rounded (default), **mitered** and **beveled** control the shape of line joins. **round** (default), **butt** and **squared** control the shape of line caps. See PostScript or PDF Reference Manual for explanation. For wild-behaving functions and thick lines it is better to use **rounded** and **round** to prevent sharp corners in line joins. (Some general support for this should be added to Gnuplot, so that the same options could be set for each line (style) separately).

dashed (default) uses different dash patterns for different line types, **solid** draws all plots with solid lines.

dashlength or **dl** scales the length of the dashed-line segments by <dl>. **linewidth** or **lw** scales all linewidths by <lw>. (lw 1 stands for 0.5bp, which is the default line width when drawing with Metapost.) **fontscale** scales text labels for factor <fontscale> relative to default document font.

mppoints uses predefined point shapes, drawn in Metapost. **texpoints** uses easily configurable set of symbols, defined with ConTeXt in the following way:

```
\defineconversion[my own points][+,{\ss x},\mathematics{\circ}]
\setupGNUPLOTterminal[context][points=tex,pointset=my own points]
```

inlineimages writes binary images to a string and only works in ConTeXt MKIV. **externalimages** writes PNG files to disk and also works with ConTeXt MKII. Gnuplot needs to have support for PNG images built in for this to work.

With **font** you can set font name and size in standalone graphics. In non-standalone (**input**) mode only the font size is important to reserve enough space for text labels. The command

```
set term context font "myfont,ss,10"
```

will result in

```
\setupbodyfont[myfont,ss,10pt]
```

If you additionally set **fontscale** to 0.8 for example, then the resulting font will be 8pt big and

```
set label ... font "myfont,12"
```

will come out as 9.6pt.

It is your own responsibility to provide proper typescripts (and header), otherwise switching the font will have no effect. For a standard font in ConTeXt MKII (pdfTeX) you could use:

```
set terminal context standalone header '\usetypescript[iwona][ec]' \
    font "iwona,ss,11"
```

Please take a look into ConTeXt documentation, wiki or mailing list (archives) for any up-to-date information about font usage.

Examples:

```
set terminal context size 10cm, 5cm     # 10cm, 5cm
set terminal context size 4in, 3in      # 4in, 3in
```

For standalone (whole-page) plots with labels in UTF-8 encoding:

```
set terminal context standalone header '\enableregime[utf-8]'
```

Requirements

You need gnuplot module for ConTeXt http://ctan.org/pkg/context-gnuplot

and a recent version of ConTeXt. If you want to call gnuplot on-the-fly, you also need write18 enabled. In most TeX distributions this can be set with shell_escape=t in texmf.cnf.

See http://wiki.contextgarden.net/Gnuplot

for details about this terminal and for more exhaustive help & examples.

Calling gnuplot from ConTeXt

The easiest way to make plots in ConTeXt documents is

```
\usemodule[gnuplot]
\starttext
\title{How to draw nice plots with {\sc gnuplot}?}
\startGNUPLOTscript[sin]
set format y "%.1f"
plot sin(x) t '$\sin(x)$'
\stopGNUPLOTscript
\useGNUPLOTgraphic[sin]
\stoptext
```

This will run gnuplot automatically and include the resulting figure in the document.

Corel

Legacy terminal for CorelDraw (circa 1995).

Syntax:

```
set terminal corel {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.

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.

Domterm

The **domterm** terminal device runs on the DomTerm terminal emulator including the domterm and qtdomterm programs. It supports SVG graphics embedded directly in the terminal output. See http://domterm.org .

Please read the help for the **svg** terminal.

Dumb

The **dumb** terminal driver plots into a text block using ascii characters. It has an optional size specification and a trailing linefeed flag.

Syntax:

```
set terminal dumb {size <xchars>,<ychars>} {[no]feed}
                  {aspect <htic>{,<vtic>}}
                  {[no]enhanced}
                  {mono|ansi|ansi256|ansirgb}
```

where <xchars> and <ychars> set the size of the text block. The default is 79 by 24. The last newline is printed only if **feed** is enabled.

The **aspect** option can be used to control the aspect ratio of the plot by setting the length of the horizontal and vertical tic marks. Only integer values are allowed. Default is 2,1 – corresponding to the aspect ratio of common screen fonts.

The **ansi**, **ansi256**, and **ansirgb** options will include escape sequences in the output to handle colors. Note that these might not be handled by your terminal. Default is **mono**. To obtain the best color match in **ansi** mode, you should use **set colorsequence classic**. Depending on the mode, the **dumb** terminal will emit the following sequences (without the additional whitespace):

```
ESC [ 0 m           reset attributes to defaults
foreground color:
ESC [ 1 m           set intense/bold
ESC [ 22 m          intense/bold off
ESC [ <fg> m        with color code 30 <= <fg> <= 37
ESC [ 39 m          reset to default
ESC [ 38; 5; <c> m  with palette index 16 <= <c> <= 255
ESC [ 38; 2; <r>; <g>; <b> m  with components 0 <= <r,g,b> <= 255
background color:
ESC [ <bg> m        with color code 40 <= <bg> <= 47
ESC [ 49 m          reset to default
ESC [ 48; 5; <c> m  with palette index 16 <= <c> <= 231
ESC [ 48; 2; <r>; <g>; <b> m  with components 0 <= <r,g,b> <= 255
```

See also e.g. the description at https://en.wikipedia.org/wiki/ANSI_escape_code#Colors

Example:

```
set term dumb mono size 60,15 aspect 1
set tics nomirror scale 0.5
plot [-5:6.5] sin(x) with impulse ls -1

    1 +-------------------------------------------------+
  0.8 +|||++                 ++||||++                  |
  0.6 +|||||+               ++|||||||+  sin(x) +-----+ |
  0.4 +||||||+             ++|||||||||+                |
  0.2 +|||||||+           ++|||||||||||+              +|
    0 ++++++++++++++++++++++++++++++++++++++++++++++++++|
 -0.2 +        +|||||||||||+             +|||||||||||+ |
 -0.4 +         +|||||||||+               +|||||||||+  |
 -0.6 +          +|||||||+                 +|||||||+   |
 -0.8 +           ++||||+                   ++||||+    |
   -1 +---+--------+--------+-------+--------+--------+-+
         -4        -2        0       2        4        6
```

Dxf

Terminal driver **dxf** for export to AutoCad (Release 10.x). It has no options. The default size is 120x80 AutoCad units. **dxf** uses seven colors (white, red, yellow, green, cyan, blue and magenta) that 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. Note: someone please update this terminal to the 2012 DXF standard!

Dxy800a

Note: legacy terminal. This terminal driver supports the Roland DXY800A plotter. It has no options.

Eepic

Note: Legacy terminal (not built by default). The latex, emtex, eepic, and tpic terminals in older versions of gnuplot provided minimal support for graphics styles beyond simple lines and points. They have been directly superseded by the **pict2e** terminal. For more capable TeX/LaTeX compatible terminal types see **cairolatex** **(p. 240)**, **context** **(p. 246)**, **epslatex** **(p. 252)**, **mp** **(p. 267)**, **pstricks** **(p. 282)**, and **tikz** **(p. 287)**.

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 DVI driver that supports the "tpic" \specials. If your driver doesn't support those \specials, "eepicemu.sty" will enable you to use some of them. dvips and dvipdfm do support the "tpic" \specials, pdflatex does not.

Syntax:

```
set terminal eepic {default} {color|monochrome|dashed}
                   {rotate} {size XX,YY}
                   {small|tiny|<fontsize>}
```

color causes gnuplot to produce \color{...} commands so that the graphs are colored. Using this option, you must include \usepackage{color} in the preamble 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. Default is to use the default math size. **tiny** uses \scriptscriptstyle symbols.

The default size of an eepic plot is 5x3 inches. You can change this using the **size** terminal option.

<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.

default resets all options to their defaults = no color, no dashed lines, pseudo-rotated (stacked) text, large point symbols.

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.)

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}}'
```

Emf

The **emf** terminal generates an Enhanced Metafile Format file. This file format is recognized by many Windows applications.

Syntax:

```
set terminal emf {color | monochrome}
                 {enhanced {noproportional}}
                 {rounded | butt}
                 {linewidth <LW>} {dashlength <DL>}
                 {size XX,YY} {background <rgb_color>}
                 {font "<fontname>{,<fontsize>}"}
                 {fontscale <scale>}
```

In **monochrome** mode successive line types cycle through dash patterns. **linewidth <factor>** multiplies all line widths by this factor. **dashlength <factor>** is useful for thick lines. <fontname> is the name of a font; and **<fontsize>** is the size of the font in points.

The nominal size of the output image defaults to 1024x768 in arbitrary units. You may specify a different nominal size using the **size** option.

Enhanced text mode tries to approximate proportional character spacing. If you are using a monospaced font, or don't like the approximation, you can turn off this correction using the **noproportional** option.

The default settings are **color font "Arial,12" size 1024,768** Selecting **default** sets all options to their default values.

Examples:

```
set terminal emf 'Times Roman Italic, 12'
```

Emxvga

Note: legacy terminal. The **emxvga** and **emxvesa** 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}
```

The only option is the vesa mode for **emxvesa**, which defaults to G640x480x256.

Epscairo

The **epscairo** terminal device generates encapsulated PostScript (*.eps) using the cairo and pango support libraries. cairo version >= 1.6 is required.

Please read the help for the **pdfcairo** terminal.

Epslatex

The **epslatex** driver generates output for further processing by LaTeX.

Syntax:

```
set terminal epslatex   {default}
set terminal epslatex   {standalone | input}
                        {oldstyle | newstyle}
                        {level1 | leveldefault | level3}
                        {color | colour | monochrome}
                        {background <rgbcolor> | nobackground}
                        {dashlength | dl <DL>}
```

```
{linewidth | lw <LW>} {pointscale | ps <PS>}
{rounded | butt}
{clip | noclip}
{palfuncparam <samples>{,<maxdeviation>}}
{size <XX>{unit},<YY>{unit}}
{header <header> | noheader}
{blacktext | colortext | colourtext}
{{font} "fontname{,fontsize}" {<fontsize>}}
{fontscale <scale>}
```

The epslatex terminal prints a plot as **terminal postscript eps** but transfers the texts to LaTeX instead of including in the PostScript code. Thus, many options are the same as in the **postscript terminal**.

The appearance of the epslatex terminal changed between versions 4.0 and 4.2 to reach better consistency with the postscript terminal: The plot size has been changed from 5 x 3 inches to 5 x 3.5 inches; the character width is now estimated to be 60% of the font size while the old epslatex terminal used 50%; now, the larger number of postscript linetypes and symbols are used. To reach an appearance that is nearly identical to the old one specify the option **oldstyle**. (In fact some small differences remain: the symbol sizes are slightly different, the tics are half as large as in the old terminal which can be changed using **set tics scale**, and the arrows have all features as in the postscript terminal.)

If you see the error message

```
"Can't find PostScript prologue file ... "
```

Please see and follow the instructions in **postscript prologue (p. 279)**.

The option **color** enables color, while **monochrome** prefers black and white drawing elements. Further, **monochrome** uses gray **palette** but it does not change color of objects specified with an explicit **colorspec**. **dashlength** or **dl** scales the length of dashed-line segments by <DL>, which is a floating-point number greater than zero. **linewidth** or **lw** scales all linewidths by <LW>.

By default the generated PostScript code uses language features that were introduced in PostScript Level 2, notably filters and pattern-fill of irregular objects such as filledcurves. PostScript Level 2 features are conditionally protected so that PostScript Level 1 interpreters do not issue errors but, rather, display a message or a PostScript Level 1 approximation. The **level1** option substitutes PostScript Level 1 approximations of these features and uses no PostScript Level 2 code. This may be required by some old printers and old versions of Adobe Illustrator. The flag **level1** can be toggled later by editing a single line in the PostScript output file to force PostScript Level 1 interpretation. In the case of files containing level 2 code, the above features will not appear or will be replaced by a note when this flag is set or when the interpreting program does not indicate that it understands level 2 PostScript or higher. The flag **level3** enables PNG encoding for bitmapped images, which can reduce the output size considerably.

rounded sets line caps and line joins to be rounded; **butt** is the default, butt caps and mitered joins.

clip tells PostScript to clip all output to the bounding box; **noclip** is the default.

palfuncparam controls how **set palette functions** are encoded as gradients in the output. Analytic color component functions (set via **set palette functions**) are encoded as linear interpolated gradients in the postscript output: The color component functions are sampled at <samples> points and all points are removed from this gradient which can be removed without changing the resulting colors by more than <maxdeviation>. For almost every useful palette you may safely leave the defaults of <samples>=2000 and <maxdeviation>=0.003 untouched.

The default size for postscript output is 10 inches x 7 inches. The default for eps output is 5 x 3.5 inches. The **size** option changes this to whatever the user requests. By default the X and Y sizes are taken to be in inches, but other units are possibly (currently only cm). The BoundingBox of the plot is correctly adjusted to contain the resized image. Screen coordinates always run from 0.0 to 1.0 along the full length of the plot edges as specified by the **size** option. NB: **this is a change from the previously recommended method of using the set size command prior to setting the terminal type**. The old method left the BoundingBox unchanged and screen coordinates did not correspond to the actual limits of the plot.

blacktext forces all text to be written in black even in color mode;

The epslatex driver offers a special way of controlling text positioning: (a) 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. (b) If the text string begins with '[', you need to continue it with: a position specification (up to two out of t,b,l,r,c), ']{', 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. See also the documentation for the **pslatex** **(p. 280)** terminal driver. To create multiline labels, use \shortstack, for example

```
set ylabel '[r]{\shortstack{first line \\ second line}}'
```

The **back** option of **set label** commands is handled slightly different than in other terminals. Labels using 'back' are printed behind all other elements of the plot while labels using 'front' are printed above everything else.

The driver produces two different files, one for the eps part of the figure and one for the LaTeX part. The name of the LaTeX file is taken from the **set output** command. The name of the eps file is derived by replacing the file extension (normally **.tex**) with **.eps** instead. There is no LaTeX output if no output file is given! Remember to close the **output file** before next plot unless in **multiplot** mode.

In your LaTeX documents use '\input{filename}' to include the figure. The **.eps** file is included by the command \includegraphics{...}, so you must also include \usepackage{graphicx} in the LaTeX preamble. If you want to use coloured text (option **textcolour**) you also have to include \usepackage{color} in the LaTeX preamble.

Pdf files can be made from the eps file using 'epstopdf'. If the graphics package is properly configured, the LaTeX files can also be processed by pdflatex without changes, using the pdf files instead of the eps files. The behaviour concerning font selection depends on the header mode. In all cases, the given font size is used for the calculation of proper spacing. When not using the **standalone** mode the actual LaTeX font and font size at the point of inclusion is taken, so use LaTeX commands for changing fonts. If you use e.g. 12pt as font size for your LaTeX document, use '"" 12' as options. The font name is ignored. If using **standalone** the given font and font size are used, see below for a detailed description.

If text is printed coloured is controlled by the TeX booleans \ifGPcolor and \ifGPblacktext. Only if \ifGPcolor is true and \ifGPblacktext is false, text is printed coloured. You may either change them in the generated TeX file or provide them globally in your TeX file, for example by using

```
\newif\ifGPblacktext
\GPblacktexttrue
```

in the preamble of your document. The local assignment is only done if no global value is given.

When using the epslatex terminal give the name of the TeX file in the **set output** command including the file extension (normally ".tex"). The eps filename is generated by replacing the extension by ".eps".

If using the **standalone** mode a complete LaTeX header is added to the LaTeX file; and "-inc" is added to the filename of the eps file. The **standalone** mode generates a TeX file that produces output with the correct size when using dvips, pdfTeX, or VTeX. The default, **input**, generates a file that has to be included into a LaTeX document using the \input command.

If a font other than "" or "default" is given it is interpreted as LaTeX font name. It contains up to three parts, separated by a comma: 'fontname,fontseries,fontshape'. If the default fontshape or fontseries are requested, they can be omitted. Thus, the real syntax for the fontname is '[fontname][,fontseries][,fontshape]'. The naming convention for all parts is given by the LaTeX font scheme. The fontname is 3 to 4 characters long and is built as follows: One character for the font vendor, two characters for the name of the font, and optionally one additional character for special fonts, e.g., 'j' for fonts with old-style numerals or 'x' for expert fonts. The names of many fonts is described in `http://www.tug.org/fontname/fontname.pdf`

For example, 'cmr' stands for Computer Modern Roman, 'ptm' for Times-Roman, and 'phv' for Helvetica. The font series denotes the thickness of the glyphs, in most cases 'm' for normal ("medium") and 'bx' or 'b' for bold fonts. The font shape is 'n' for upright, 'it' for italics, 'sl' for slanted, or 'sc' for small caps, in general. Some fonts may provide different font series or shapes.

Examples:

Use Times-Roman boldface (with the same shape as in the surrounding text):

```
set terminal epslatex 'ptm,bx'
```

Use Helvetica, boldface, italics:

```
set terminal epslatex 'phv,bx,it'
```

Continue to use the surrounding font in slanted shape:

```
set terminal epslatex ',,sl'
```

Use small capitals:

```
set terminal epslatex ',,sc'
```

By this method, only text fonts are changed. If you also want to change the math fonts you have to use the "gnuplot.cfg" file or the **header** option, described below.

In standalone mode, the font size is taken from the given font size in the **set terminal** command. To be able to use a specified font size, a file "size<size>.clo" has to reside in the LaTeX search path. By default, 10pt, 11pt, and 12pt are supported. If the package "extsizes" is installed, 8pt, 9pt, 14pt, 17pt, and 20pt are added.

The **header** option takes a string as argument. This string is written into the generated LaTeX file. If using the **standalone** mode, it is written into the preamble, directly before the \begin{document} command. In the **input** mode, it is placed directly after the \begingroup command to ensure that all settings are local to the plot.

Examples:

Use T1 fontencoding, change the text and math font to Times-Roman as well as the sans-serif font to Helvetica:

```
set terminal epslatex standalone header \
"\\usepackage[T1]{fontenc}\n\\usepackage{mathptmx}\n\\usepackage{helvet}"
```

Use a boldface font in the plot, not influencing the text outside the plot:

```
set terminal epslatex input header "\\bfseries"
```

If the file "gnuplot.cfg" is found by LaTeX it is input in the preamble the LaTeX document, when using **standalone** mode. It can be used for further settings, e.g., changing the document font to Times-Roman, Helvetica, and Courier, including math fonts (handled by "mathptmx.sty"):

```
\usepackage{mathptmx}
\usepackage[scaled=0.92]{helvet}
\usepackage{courier}
```

The file "gnuplot.cfg" is loaded before the header information given by the **header** command. Thus, you can use **header** to overwrite some of settings performed using "gnuplot.cfg"

Epson_180dpi

Note: only available if gnuplot is configured –with-bitmap-terminals. 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.

The **dpu414** driver is for the Seiko DPU-414 thermal printer.

nec_cp6 has the options:

Syntax:

```
set terminal nec_cp6 {monochrome | colour | draft}
```

which defaults to monochrome.

dpu414 has the options:

Syntax:

```
set terminal dpu414 {small | medium | large} {normal | draft}
```

which defaults to medium (=font size) and normal. Preferred combinations are **medium normal** and **small draft**.

Excl

Note: legacy terminal. The **excl** terminal driver supports Talaris printers such as the EXCL Laser printer and the 1590. It has no options.

Fig

The **fig** terminal device generates output in the Fig graphics language for import into the xfig interactive drawing tool. Notes:

```
The fig terminal was significantly revised in gnuplot version 5.3.
Currently only version 3.2 of the fig file format is supported.
Use of dash patterns may require Xfig 3.2.6 or newer.
```

Syntax:

```
set terminal fig {monochrome | color}
                 {small | big | size <xsize>{in|cm},<ysize>{in|cm}}
                 {landscape | portrait}
                 {font "<fontname>{,<fontsize>}"} {fontsize <size>}
                 {textnormal | {textspecial texthidden textrigid}}
                 {{linewidth|lw} <multiplier>}
```

The default settings are

```
set term fig color small landscape font "Times Roman,10" lw 1.0
```

size sets the size of the drawing area to <xsize>*<ysize> in units of inches (default) or centimeters. The default is **size 5in,3in**. **small** is shorthand for **size 5in,3in** (3in,5in in portrait mode). **big** is shorthand for **size 8in,5in**.

font sets the text font face to <fontname> and its size to <fontsize> points. Choice is limited to the 35 standard PostScript fonts. **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.

linewidth is a multiplier for the linewidth property of all lines.

Additional point-plot symbols are also available in 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 scales with the font size.

RGB colors will be replaced with gray unless they have been defined in a linetype prior to plotting or match a known named color or palette value. See **colornames** **(p. 139)**. E.g.

```
set linetype 999 lc rgb '#aabbcc'
plot $data with fillecurve fillcolor rgb '#aabbcc'
```

Ggi

Legacy terminal driver for the GGI (General Graphics Interface) project. 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
```

Gif

Syntax:

```
set terminal gif
       {{no}enhanced}
       {{no}transparent} {rounded|butt}
       {linewidth <lw>} {dashlength <dl>}
       {tiny | small | medium | large | giant}
       {font "<face> {,<pointsize>}"} {fontscale <scale>}
       {size <x>,<y>} {{no}crop}
       {background <rgb_color>}
       {animate {delay <d>} {loop <n>} {optimize}}
```

PNG, JPEG and GIF images are created using the external library libgd. GIF plots may be viewed interactively by piping the output to the 'display' program from the ImageMagick package as follows:

```
set term gif
set output '| display gif:-'
```

You can view the output from successive plot commands interactively by typing <space> in the display window. To save the current plot to a file, left click in the display window and choose **save**.

transparent instructs the driver to make the background color transparent. Default is **notransparent**.

The **linewidth** and **dashlength** options are scaling factors that affect all lines drawn, i.e. they are multiplied by values requested in various drawing commands.

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**.

The output plot size <x,y> is given in pixels — it defaults to 640x480. Please see additional information under **canvas (p. 31)** and **set size (p. 195)**. Blank space at the edges of the finished plot may be trimmed using the **crop** option, resulting in a smaller final image size. Default is **nocrop**.

Animate

```
set term gif animate {delay <d>} {loop <n>} {{no}optimize}}
```

The gif terminal **animate** option creates a single gif file containing multiple frames. The delay between display of successive frames may be specified in units of 1/100 second (default 5), but this value may or may not be honored accurately by a program used to view the animation later. The number of animation loops during playback can be specified, with the default of 0 meaning unlimited looping. Again this value may or may not be honored by the program later used for viewing. An animation sequence is terminated by the next **set output** or **set term** command.

The **optimize** option [DEPRECATED] is passed to the gd library when the output file is opened. It has two effects on the animation.

1) A single color map is used for the entire animation. This requires that all colors used in any frame of the animation are already defined in the first frame.

2) If possible, only the portions of a frame that differ from the previous frame are stored in the animation file. This space saving may not be possible if the animation uses transparency.

Both of these optimizations are intended to produce a smaller output file, but the decrease in size is probably only significant for long animations. Caveat: The implementation of optimization in libgd is known to be buggy. Therefore use of this option in gnuplot is not recommended.

Example showing continuous rotation:

```
set term gif animate loop 0
set output 'rotating_surface.gif'
do for [ang=1:359] {
    set view 60, ang
    splot f(x,y) with pm3d
}
unset output
```

Fonts

The details of font selection are complicated. For more information please see the separate section under **fonts gd (p. 45)**.

Examples:

```
set terminal gif medium noenhanced size 640,480 background '#ffffff'
```

Use the medium size built-in non-scaleable, non-rotatable font. Enhanced text mode will not work with this font. Use white (24 bit RGB in hexadecimal) for the non-transparent background.

```
set terminal gif font arial 14
```

Searches for a font with face name 'arial' and sets the font size to 14pt.

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 **set label** 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 specifying x and y in the command

```
set terminal gpic x y
```

Grass

Note: legacy terminal. 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.

Hp2623a

The **hp2623a** terminal driver supports the Hewlett Packard HP2623A. It has no options.

Hp2648

The **hp2648** terminal driver supports the Hewlett Packard HP2647 and HP2648. It has no options.

Hp500c

Note: only available if gnuplot is configured –with-bitmap-terminals. 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.

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 **set encoding (p. 146)** 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. HPGL graphics can be imported by many software packages.

Hpljii

Note: only available if gnuplot is configured –with-bitmap-terminals. 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.

Hppj

Note: only available if gnuplot is configured –with-bitmap-terminals. 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.

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.

Jpeg

Syntax:

```
set terminal jpeg
       {{no}enhanced}
       {{no}interlace}
       {linewidth <lw>} {dashlength <dl>} {rounded|butt}
       {tiny | small | medium | large | giant}
       {font "<face> {,<pointsize>}"} {fontscale <scale>}
       {size <x>,<y>} {{no}crop}
       {background <rgb_color>}
```

PNG, JPEG and GIF images are created using the external library libgd. In most cases, PNG is to be preferred for single plots, and GIF for animations. Both are loss-less image formats, and produce better image quality than the lossy JPEG format. This is in particular noticeable for solid color lines against a solid background, i.e. exactly the sort of image typically created by gnuplot.

The **interlace** option creates a progressive JPEG image. Default is **nointerlace**.

The **linewidth** and **dashlength** options are scaling factors that affect all lines drawn, i.e. they are multiplied by values requested in various drawing commands.

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**.

The details of font selection are complicated. Two equivalent simple examples are given below:

```
set term jpeg font arial 11
set term jpeg font "arial,11"
```

For more information please see the separate section under **fonts** **(p. 45)**.

The output plot size <x,y> is given in pixels — it defaults to 640x480. Please see additional information under **canvas** **(p. 31)** and **set size** **(p. 195)**. Blank space at the edges of the finished plot may be trimmed using the **crop** option, resulting in a smaller final image size. Default is **nocrop**.

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.

Latex

Note: Legacy terminal (not built by default). The latex, emtex, eepic, and tpic terminals in older versions of gnuplot provided minimal support for graphics styles beyond simple lines and points. They have been directly superseded by the **pict2e** terminal. For more capable TeX/LaTeX compatible terminal types see **cairolatex** **(p. 240)**, **context** **(p. 246)**, **epslatex** **(p. 252)**, **mp** **(p. 267)**, **pstricks** **(p. 282)**, and **tikz** **(p. 287)**.

Syntax:

```
set terminal {latex | emtex} {default | {courier|roman} {<fontsize>}}
             {size <XX>{unit}, <YY>{unit}} {rotate | norotate}
             {color | monochrome}
```

By default the plot will inherit font settings from the embedding document. You have the option of forcing either Courier (cmtt) or Roman (cmr) fonts instead. In this case you may also specify a fontsize. 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. Other point types use symbols from the amssymb package.

The default size for the plot is 5 inches by 3 inches. The **size** option changes this to whatever the user requests. By default the X and Y sizes are taken to be in inches, but other units are possible (currently only cm).

If **rotate** is specified, rotated text, especially a rotated y-axis label, is possible (the packages graphics or graphicx are needed). The 'stacked' y-axis label mechanism is then deactivated. This will also significantly improve the quality of line drawing, and is default since version 5.3.

The option **color** enables color, while **monochrome** uses only black and white drawing elements. You need to load the color or xcolor package in the preamble of your latex document.

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}}'
```

Linux console

Older gnuplot versions required special terminals **linux** or **vgagl** in order to display graphics on the linux console, i.e. in the absence of X11 or other windowing environment. These terminals have been deprecated.

The recommended way to run gnuplot from the linux console is now to use a console terminal emulator such as yaft (https://github.com/uobikiemukot/yaft) that supports sixel graphics. With yaft as your console terminal you can run gnuplot and select a terminal with sixel output. See **sixelgd (p. 284)**. As a fall-back option you could use **set term dumb**, but sixel graphics are much nicer.

Lua

The **lua** generic terminal driver works in conjunction with an external Lua script to create a target-specific plot file. Currently the only supported target is TikZ -> pdflatex.

Information about Lua is available at http://www.lua.org .

Syntax:

```
set terminal lua <target name> | "<file name>"
                    {<script_args> ...}
                    {help}
```

A 'target name' or 'file name' (in quotes) for a script is mandatory. If a 'target name' for the script is given, the terminal will look for "gnuplot-<target name>.lua" in the local directory and on failure in the environmental variable GNUPLOT_LUA_DIR.

All arguments will be provided to the selected script for further evaluation. E.g. 'set term lua tikz help' will cause the script itself to print additional help on options and choices for the script.

Lua tikz

The TikZ driver is one output mode of the generic Lua terminal.

Syntax:

```
 set terminal lua tikz

  {latex | tex | context}
  {color | monochrome}
  {nooriginreset | originreset}
  {nogparrows | gparrows}
  {nogppoints | gppoints}
  {picenvironment | nopicenvironment}
  {noclip | clip}
  {butt}
  {notightboundingbox | tightboundingbox}
  {background "<colorpec>"}
  {size <x>{unit},<y>{unit}}
  {scale <x>,<y>}
  {plotsize <x>{unit},<y>{unit}}
  {charsize <x>{unit},<y>{unit}}
```

```
{font "<fontdesc>"}
{{fontscale | textscale} <scale>}
{dashlength | dl <DL>}
{linewidth | lw <LW>}
{nofulldoc | nostandalone | fulldoc | standalone}
{{preamble | header} "<preamble_string>"}
{tikzplot <ltn>,...}
{notikzarrows | tikzarrows}
{rgbimages | cmykimages}
{noexternalimages|externalimages}
{bitmap | nobitmap}
{providevars <var name>,...}
{createstyle}
{help}
```

For all options that expect lengths as their arguments they will default to 'cm' if no unit is specified. For all lengths the following units may be used: 'cm', 'mm', 'in' or 'inch', 'pt', 'pc', 'bp', 'dd', 'cc'. Blanks between numbers and units are not allowed.

'monochrome' disables line coloring and switches to grayscaled fills.

'originreset' moves the origin of the TikZ picture to the lower left corner of the plot. It may be used to align several plots within one tikzpicture environment. This is not tested with multiplots and pm3d plots!

'gparrows' use gnuplot's internal arrow drawing function instead of the ones provided by TikZ.

'gppoints' use gnuplot's internal plotmark drawing function instead of the ones provided by TikZ.

'nopicenvironment' omits the declaration of the 'tikzpicture' environment in order to set it manually. This permits putting some PGF/TikZ code directly before or after the plot.

'clip' crops the plot at the defined canvas size. Default is 'noclip' by which only a minimum bounding box of the canvas size is set. Neither a fixed bounding box nor a crop box is set if the 'plotsize' or 'tightboundingbox' option is used.

'butt' changes the linecap property to "butt" and the linejoin property to "miter". The defaults are "round" and "round".

If 'tightboundingbox' is set the 'clip' option is ignored and the final bounding box is the natural bounding box calculated by tikz.

'background' sets the background color to the value specified in the <colorpec> argument. <colorspec> must be a valid color name or a 3 byte RGB code as a hexadecimal number with a preceding number sign ('#'). E.g. '#ff0000' specifies pure red. If omitted the background is transparent.

The 'size' option expects two lengths <x> and <y> as the canvas size. The default size of the canvas is 12.5cm x 8.75cm.

The 'scale' option works similar to the 'size' option but expects scaling factors <x> and <y> instead of lengths.

The 'plotsize' option permits setting the size of the plot area instead of the canvas size, which is the usual gnuplot behaviour. Using this option may lead to slightly asymmetric tic lengths. Like 'originreset' this option may not lead to convenient results if used with multiplots or pm3d plots. An alternative approach is to set all margins to zero and to use the 'noclip' option. The plot area has then the dimensions of the given canvas sizes.

The 'charsize' option expects the average horizontal and vertical size of the used font. Look at the generated style file for an example of how to use it from within your TeX document.

'fontscale' or 'textscale' expects a scaling factor as a parameter. All texts in the plot are scaled by this factor then.

'dashlength' or 'dl' scales the length of dashed-line segments by <DL>, which is a floating-point number greater than zero. 'linewidth' or 'lw' scales all linewidths by <LW>.

The options 'tex', 'latex' and 'context' choose the TeX output format. LaTeX is the default. To load the style file put the according line at the beginning of your document:

```
\input gnuplot-lua-tikz.tex    % (for plain TeX)
\usepackage{gnuplot-lua-tikz}  % (for LaTeX)
\usemodule[gnuplot-lua-tikz]   % (for ConTeXt)
```

'createstyle' derives the TeX/LaTeX/ConTeXt styles from the script and writes them to the appropriate files.

'fulldoc' or 'standalone' produces a full LaTeX document for direct compilation.

'preamble' or 'header' may be used to put any additional LaTeX code into the document preamble in standalone mode.

With the 'tikzplot' option the '\path plot' command will be used instead of only '\path'. The following list of numbers of linetypes (<ltn>,...) defines the affected plotlines. There exists a plotstyle for every linetype. The default plotstyle is 'smooth' for every linetype >= 1.

By default the tikz terminal produces simple LaTeX arrows. To produce arrows in accord with gnuplot's 'arrowstyle' settings, use the 'gparrows' option. The 'tikzarrows' option is a third alternative that bypasses both of these. Instead the arrowstyle 'angle' parameter is used to index a set of 12 pre-defined TikZ arrow styles. E.g. an arrow style with the angle '7' will be mapped to the TikZ style 'gp arrow 7' ignoring all other arrowstyle settings.

With 'cmykimages' the CMYK color model will be used for inline image data instead of the RGB model. All other colors (like line colors etc.) are not affected by this option, since they are handled e.g. by LaTeX's xcolor package. This option is ignored if images are externalized.

By using the 'externalimages' option all bitmap images will be written as external PNG images and included at compile time of the document. Generating DVI and later postscript files requires to convert the PNGs into EPS files in a separate step e.g. by using ImageMagick's **convert**. Transparent bitmap images are always generated as an external PNGs.

The 'nobitmap' option let images be rendered as filled rectangles instead of the nativ PS or PDF inline image format. This option is ignored if images are externalized.

The 'providevars' options makes gnuplot's internal and user variables available by using the '\gpgetvar{<var name>}' command within the TeX script. Use gnuplot's 'show variables all' command to see the list of valid variables.

The <fontdesc> string may contain any valid TeX/LaTeX/ConTeXt font commands like e.g. '\small'. It is passed directly as a node parameter in form of "font={<fontdesc>}". This can be 'misused' to add further code to a node, e.g. '\small,yshift=1ex' or ',yshift=1ex' are also valid while the latter does not change the current font settings. One exception is the second argument of the list. If it is a number of the form <number>{unit} it will be interpreted as a fontsize like in other terminals and will be appended to the first argument. If the unit is omitted the value is interpreted as 'pt'. As an example the string '\sffamily,12,fill=red' sets the font to LaTeX's sans serif font at a size of 12pt and red background color. The same applies to ConTeXt, e.g. '\switchtobodyfont[iwona],10' changes the font to Iwona at a size of 10pt. Plain TeX users have to change the font size explicitly within the first argument. The second should be set to the same value to get proper scaling of text boxes.

Strings have to be put in single or double quotes. Double quoted strings may contain special characters like newlines '\n' etc.

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.

Mif

Note: Legacy terminal. 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} {polyline | 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. **help** and **?** print online help on standard error output — both print a short description of the usage; **help** also lists the options.

Examples:

```
      set term mif colour polylines    # defaults
      set term mif                     # defaults
      set term mif vectors
      set term mif help
```

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 **set label** 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.

Pbm

Note: only available if gnuplot is configured –with-bitmap-terminals. 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 size is white-space padded to the nearest multiple of 8 pixels on both x and y. This empty space may be cropped later if needed.

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 | pnmtopng > tilted.png'  # uses NETPBM
```

Pcl5

The **pcl5** driver supports PCL5e/PCL5c printers. It (mostly) uses the HP-GL/2 vector format.

Syntax:

```
set terminal pcl5 {<mode>} {{no}enhanced}
    {size <plotsize> | size <width>{unit},<height>{unit}}
    {font "<fontname>,<size>"} {pspoints | nopspoints}
    {fontscale <scale>} {pointsize <scale>} {linewidth <scale}
    {rounded|butt} {color <number_of_pens>}
```

<mode> is **landscape** or **portrait**. <plotsize> is the physical plotting size of the plot, which can be one of the following formats: **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), **extended** for (36" X 55") displays (almost a legal size ratio), or **a4** for (296mm X 210mm) displays. You can also explicitly specify the canvas size using the **width** and **height** options. Default unit is **in**. Default size is **letter**.

<fontname> can be one of stick, univers (default), albertus, antique_olive, arial, avant_garde_gothic, bookman, zapf_chancery, clarendon, coronet, courier courier_ps, cg_times, garamond_antigua, helvetica, helvetica_narrow, letter_gothic, marigold, new_century_schlbk, cg_omega, palatino, times_new_roman, times_roman, zapf_dingbats, truetype_symbols, or wingdings. Font names are case-insensitive and underscores may be replaced by spaces or dashes or may be left out. <fontsize> is the font size in points.

The point type selection can be the a limited default set by specifying **nopspoints**, or the same set of point types as provided by the postscript terminal by specifying **pspoints** (default).

The **butt** option selects lines with butt ends and mitered joins (default), whereas **rounded** selects rounded line ends and joins.

Line widths, and point and font sizes can be scaled using the **linewidth**, **pointscale**, or **fontscale** options, respectively. **color** selects the number of pens <number_of_pens> used in plots. Default is 8, minimum 2.

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 monochrome devices, but newer ones will do grey-scale.

Defaults: landscape, a4, 8 pens, univers, 12 point, pspoints, butt, no scaling

The **pcl5** terminal will try to request fonts which match your **encoding**. Note that this has highest priority, so you might end up with a different font face. The terminal's default **encoding** is **HP Roman-8**.

Limitations:

This terminal does not support alpha transparency. Transparent filling is emulated using shading patterns. Boxed text is not implemented.

The support for UTF-8 is limited. Lacking the label mode for UTF-8 output in HP-GL/2, the driver reverts to PCL for strings containing 8bit characters. UTF-8 text is limited to angles of 0, 90, 180, and 270 degrees. Also vertical alignment might be off depending on the font.

Some enhanced text features (phantom box, overprinting) require using PCL features in addition to HP-GL/2. This conforms to the specs but may not work with your printer or software.

Pdf

[DEPRECATED] This terminal uses the non-free library PDFlib (GmbH Munchen) to produce files in Portable Document Format. Unless you have a commercial license for PDFlib and need some special feature it provides you would do better to use the cairo pdf terminal instead. Gnuplot can also export PDF files from wxt or qt interactive terminal sessions.

Syntax:

```
set terminal pdf {monochrome|color|colour}
                 {{no}enhanced}
                 {fname "<font>"} {fsize <fontsize>}
                 {font "<fontname>{,<fontsize>}"} {fontscale <scale>}
                 {linewidth <lw>} {rounded|butt}
                 {dl <dashlength>}}
                 {size <XX>{unit},<YY>{unit}}
```

The default is to use a different color for each line type. Selecting **monochome** will use black for all linetypes, 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 (p. 34)**.

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.

The default size for PDF output is 5 inches by 3 inches. The **size** option changes this to whatever the user requests. By default the X and Y sizes are taken to be in inches, but other units are possible (currently only cm).

Pdfcairo

The **pdfcairo** terminal device generates output in pdf. The actual drawing is done via cairo, a 2D graphics library, and pango, a library for laying out and rendering text.

Syntax:

```
set term pdfcairo
             {{no}enhanced} {mono|color}
             {font <font>} {fontscale <scale>}
```

```
{linewidth <lw>} {rounded|butt|square} {dashlength <dl>}
{background <rgbcolor>}
{size <XX>{unit},<YY>{unit}}
```

This terminal supports an enhanced text mode, which allows font and other formatting commands (subscripts, superscripts, etc.) to be embedded in labels and other text strings. The enhanced text mode syntax is shared with other gnuplot terminal types. See **enhanced (p. 34)** for more details.

The width of all lines in the plot can be modified by the factor <lw> specified in **linewidth**. The default linewidth is 0.5 points. (1 "PostScript" point = 1/72 inch = 0.353 mm)

rounded sets line caps and line joins to be rounded; **butt** is the default, butt caps and mitered joins.

The default size for the output is 5 inches x 3 inches. The **size** option changes this to whatever the user requests. By default the X and Y sizes are taken to be in inches, but other units are possible (currently only cm). Screen coordinates always run from 0.0 to 1.0 along the full length of the plot edges as specified by the **size** option.

<font> is in the format "FontFace,FontSize", i.e. the face and the size comma-separated in a single string. FontFace is a usual font face name, such as 'Arial'. If you do not provide FontFace, the pdfcairo terminal will use 'Sans'. FontSize is the font size, in points. If you do not provide it, the pdfcairo terminal will use a nominal font size of 12 points. However, the default fontscale parameter for this terminal is 0.5, so the apparent font size is smaller than this if the pdf output is viewed at full size.

```
For example :
   set term pdfcairo font "Arial,12"
   set term pdfcairo font "Arial" # to change the font face only
   set term pdfcairo font ",12" # to change the font size only
   set term pdfcairo font "" # to reset the font name and size
```

The fonts are retrieved from the usual fonts subsystems. Under Windows, those fonts are to be found and configured in the entry "Fonts" of the control panel. Under UNIX, they are handled by "fontconfig".

Pango, the library used to layout the text, is based on utf-8. Thus, the pdfcairo terminal has to convert from your encoding to utf-8. The default input encoding is based on your 'locale'. If you want to use another encoding, make sure gnuplot knows which one you are using. See **encoding (p. 146)** for more details.

Pango may give unexpected results with fonts that do not respect the unicode mapping. With the Symbol font, for example, the pdfcairo terminal will use the map provided by http://www.unicode.org/ to translate character codes to unicode. Note that "the Symbol font" is to be understood as the Adobe Symbol font, distributed with Acrobat Reader as "SY______.PFB". Alternatively, the OpenSymbol font, distributed with OpenOffice.org as "opens___.ttf", offers the same characters. Microsoft has distributed a Symbol font ("symbol.ttf"), but it has a different character set with several missing or moved mathematic characters. If you experience problems with your default setup (if the demo enhancedtext.dem is not displayed properly for example), you probably have to install one of the Adobe or OpenOffice Symbol fonts, and remove the Microsoft one. Other non-conform fonts, such as "wingdings" have been observed working.

The rendering of the plot cannot be altered yet. To obtain the best output possible, the rendering involves two mechanisms : antialiasing and oversampling. Antialiasing allows to display non-horizontal and non-vertical lines smoother. Oversampling combined with antialiasing provides subpixel accuracy, so that gnuplot can draw a line from non-integer coordinates. This avoids wobbling effects on diagonal lines ('plot x' for example).

Pict2e

The **pict2e** terminal uses the LaTeX2e variant of the picture environment. It replaces terminals which were based on the original LaTeX picture environment: **latex**, **emtex**, **tpic**, and **eepic**. (EXPERIMENTAL)

Alternatives to this terminal with a more complete support of gnuplot's features are **tikz**, **pstricks**, **cairolatex**, **pslatex**, **epslatex** and **mp**.

Syntax:

```
set terminal pict2e
             {font "{<fontname>}{,<fontsize>}"}
             {size <XX>{unit}, <YY>{unit}}
             {color | monochrome}
             {linewidth <lw>} {rounded | butt}
             {texarrows | gparrows} {texpoints | gppoints}
             {smallpoints | tinypoints | normalpoints}
```

This terminal requires the following standard LaTeX packages: **pict2e**, **xcolor**, **graphics/graphicx** and **amssymb**. For pdflatex, the **transparent** package is used to support transparency.

By default the plot will inherit font settings from the embedding document. You have the option to force a font with the **font** option, like cmtt (Courier) or cmr (Roman), instead. In this case you may also force a specific fontsize. Otherwise the fontsize argument is used to estimate the required space for text. Unless your driver is capable of building fonts at any size (e.g. dvips), stick to the standard 10, 11 and 12 point sizes.

The default size for the plot is 5 inches by 3 inches. The **size** option changes this to whatever the user requests. By default the X and Y sizes are taken to be in inches, but other units are possible (currently only cm).

With **texpoints**, points are drawn using LaTeX commands like "\Diamond" and "\Box". These are provided by the the latexsym package, which is part of the base distribution and thus part of any LaTeX implementation. Other point types use symbols from the amssymb package. With **gppoints**, the terminal will use gnuplot's internal routines for drawing point symbols instead.

With the **texpoints** option, you can select three different point sizes: **normalpoints**, **smallpoints**, and **tinypoints**.

color causes gnuplot to produce \color{...} commands so that the graphs are colored. Using this option, you must include \usepackage{xcolor} in the preamble of your LaTeX document. **monochrome** will avoid the use of any color commands in the output. Transparent color fill is available if pdflatex is used.

linewidth sets the scale factor for the width of lines. **rounded** sets line caps and line joins to be rounded. **butt** sets butt caps and mitered joins and is the default.

pict2e only supports dotted lines, but not dashed lines. All default line types are solid. Use **set linetype** with the **dashtype** property to change.

texarrows draws **arrow**s using LaTeX commands which are shorter but do not offer all options. **gparrows** selects drawing arrows using gnuplot's own routine for full functionality instead.

Pm

The **pm** terminal driver provides an OS/2 Presentation Manager window in which the graph is plotted. The window is opened when the first graph is plotted. This window has its own online help as well as facilities for printing and copying to the clipboard.

Syntax:

```
set terminal pm      {{server} {n} | noserver}
                     {nopersist | persist}
                     {enhanced | noenhanced}
                     {font <fontspec>}
                     {nowidelines | widelines}
                     {fontscale <scale>}
                     {linewidth <scale>}
                     {pointscale <scale>}
                     {{title} "title"}
```

If **persist** is specified, each graph appears in its own window and all windows remain open after **gnuplot** exits. If **server** is specified, all graphs appear in the same window, which remains open when **gnuplot** exits. This option takes an optional numerical argument which specifies an instance of the server process. Thus multiple server windows can be in use at the same time.

If **widelines** is specified, all plots will be drawn with wide lines. If **enhanced** is specified, sub- and superscripts and multiple fonts are enabled (see **enhanced text (p. 34)** for details). Font names for the core PostScript fonts may be abbreviated to a single letter (T/H/C/S for Times/Helvetica/Courier/Symbol).

linewidth, **fontscale**, **pointscale** can be used to scale the width of lines, the size of text, or the size of the point symbols.

If **title** is specified, it will be used as the title of the plot window. It will also be used as the name of the server instance, and will override the optional numerical argument.

The gnuplot outboard driver, gnupmdrv.exe, is searched in the same directory as gnuplot itself. You can override that by defining one of the environment variables GNUPLOT_DRIVER_DIR or GNUPLOT. As a last resort the current directory and the PATH are tried to locate gnupmdrv.exe.

Png

Syntax:

```
set terminal png
       {{no}enhanced}
       {{no}transparent} {{no}interlace}
       {{no}truecolor} {rounded|butt}
       {linewidth <lw>} {dashlength <dl>}
       {tiny | small | medium | large | giant}
       {font "<face> {,<pointsize>}"} {fontscale <scale>}
       {size <x>,<y>} {{no}crop}
       {background <rgb_color>}
```

PNG, JPEG and GIF images are created using the external library libgd. PNG plots may be viewed interactively by piping the output to the 'display' program from the ImageMagick package as follows:

```
set term png
set output '| display png:-'
```

You can view the output from successive plot commands interactively by typing <space> in the display window. To save the current plot to a file, left click in the display window and choose **save**.

transparent instructs the driver to make the background color transparent. Default is **notransparent**.

interlace instructs the driver to generate interlaced PNGs. Default is **nointerlace**.

The **linewidth** and **dashlength** options are scaling factors that affect all lines drawn, i.e. they are multiplied by values requested in various drawing commands.

By default output png images use 256 indexed colors. The **truecolor** option instead creates TrueColor images with 24 bits of color information per pixel. Transparent fill styles require the **truecolor** option. See **fillstyle (p. 199)**. A transparent background is possible in either indexed or TrueColor images.

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**.

The details of font selection are complicated. Two equivalent simple examples are given below:

```
set term png font arial 11
set term png font "arial,11"
```

For more information please see the separate section under **fonts (p. 45)**.

The output plot size <x,y> is given in pixels — it defaults to 640x480. Please see additional information under **canvas (p. 31)** and **set size (p. 195)**. Blank space at the edges of the finished plot may be trimmed using the **crop** option, resulting in a smaller final image size. Default is **nocrop**.

Examples

```
set terminal png medium size 640,480 background '#ffffff'
```

Use the medium size built-in non-scaleable, non-rotatable font. Use white (24-bit RGB in hexadecimal) for the non-transparent background.

```
set terminal png font arial 14 size 800,600
```

Searches for a scalable font with face name 'arial' and sets the font size to 14pt. Please see **fonts (p. 45)** for details of how the font search is done.

```
set terminal png transparent truecolor enhanced
```

Use 24 bits of color information per pixel, with a transparent background. Use the **enhanced text** mode to control the layout of strings to be printed.

Pngcairo

The **pngcairo** terminal device generates output in png. The actual drawing is done via cairo, a 2D graphics library, and pango, a library for laying out and rendering text.

Syntax:

```
set term pngcairo
             {{no}enhanced} {mono|color}
             {{no}transparent} {{no}crop} {background <rgbcolor>
             {font <font>} {fontscale <scale>}
             {linewidth <lw>} {rounded|butt|square} {dashlength <dl>}
             {pointscale <ps>}
             {size <XX>{unit},<YY>{unit}}
```

This terminal supports an enhanced text mode, which allows font and other formatting commands (subscripts, superscripts, etc.) to be embedded in labels and other text strings. The enhanced text mode syntax is shared with other gnuplot terminal types. See **enhanced (p. 34)** for more details.

The width of all lines in the plot can be modified by the factor <lw>.

rounded sets line caps and line joins to be rounded; **butt** is the default, butt caps and mitered joins.

The default size for the output is 640 x 480 pixels. The **size** option changes this to whatever the user requests. By default the X and Y sizes are taken to be in pixels, but other units are possible (currently cm and inch). A size given in centimeters or inches will be converted into pixels assuming a resolution of 72 dpi. Screen coordinates always run from 0.0 to 1.0 along the full length of the plot edges as specified by the **size** option.

<font> is in the format "FontFace,FontSize", i.e. the face and the size comma-separated in a single string. FontFace is a usual font face name, such as 'Arial'. If you do not provide FontFace, the pngcairo terminal will use 'Sans'. FontSize is the font size, in points. If you do not provide it, the pngcairo terminal will use a size of 12 points.

```
For example :
   set term pngcairo font "Arial,12"
   set term pngcairo font "Arial" # to change the font face only
   set term pngcairo font ",12" # to change the font size only
   set term pngcairo font "" # to reset the font name and size
```

The fonts are retrieved from the usual fonts subsystems. Under Windows, those fonts are to be found and configured in the entry "Fonts" of the control panel. Under UNIX, they are handled by "fontconfig".

Pango, the library used to layout the text, is based on utf-8. Thus, the pngcairo terminal has to convert from your encoding to utf-8. The default input encoding is based on your 'locale'. If you want to use another encoding, make sure gnuplot knows which one you are using. See **encoding (p. 146)** for more detail.

Pango may give unexpected results with fonts that do not respect the unicode mapping. With the Symbol font, for example, the pngcairo terminal will use the map provided by http://www.unicode.org/ to translate character codes to unicode. Note that "the Symbol font" is to be understood as the Adobe Symbol font, distributed with Acrobat Reader as "SY______.PFB". Alternatively, the OpenSymbol font, distributed with OpenOffice.org as "opens___.ttf", offers the same characters. Microsoft has distributed a Symbol font ("symbol.ttf"), but it has a different character set with several missing or moved mathematic characters. If you experience problems with your default setup (if the demo enhancedtext.dem is not displayed properly for example), you probably have to install one of the Adobe or OpenOffice Symbol fonts, and remove the Microsoft one.

Rendering uses oversampling, antialiasing, and font hinting to the extent supported by the cairo and pango libraries.

Postscript

Several options may be set in the **postscript** driver.

Syntax:

```
set terminal postscript {default}
set terminal postscript {landscape | portrait | eps}
                        {enhanced | noenhanced}
                        {defaultplex | simplex | duplex}
                        {fontfile {add | delete} "<filename>"
                         | nofontfiles} {{no}adobeglyphnames}
                        {level1 | leveldefault | level3}
                        {color | colour | monochrome}
                        {background <rgbcolor> | nobackground}
                        {dashlength | dl <DL>}
                        {linewidth | lw <LW>} {pointscale | ps <PS>}
                        {rounded | butt}
                        {clip | noclip}
                        {palfuncparam <samples>{,<maxdeviation>}}
                        {size <XX>{unit},<YY>{unit}}
                        {blacktext | colortext | colourtext}
                        {{font} "fontname{,fontsize}" {<fontsize>}}
                        {fontscale <scale>}
```

If you see the error message

```
"Can't find PostScript prologue file ... "
```

Please see and follow the instructions in **postscript prologue (p. 279)**.

landscape and **portrait** choose the plot orientation. **eps** mode generates EPS (Encapsulated PostScript) output, which is just regular PostScript with some additional lines that allow the file to be imported into a variety of other applications. (The added lines are PostScript comment lines, so the file may still be printed by itself.) To get EPS output, use the **eps** mode and make only one plot per file. In **eps** mode the whole plot, including the fonts, is reduced to half of the default size.

enhanced enables enhanced text mode features (subscripts, superscripts and mixed fonts). See **enhanced (p. 34)** for more information. **blacktext** forces all text to be written in black even in color mode;

Duplexing in PostScript is the ability of the printer to print on both sides of the same sheet of paper. With **defaultplex**, the default setting of the printer is used; with **simplex** only one side is printed; **duplex** prints on both sides (ignored if your printer can't do it).

"<**fontname**>" is the name of a valid PostScript font; and <**fontsize**> is the size of the font in PostScript points. In addition to the standard postscript fonts, an oblique version of the Symbol font, useful for mathematics, is defined. It is called "Symbol-Oblique".

default sets all options to their defaults: **landscape**, **monochrome**, **dl 1.0**, **lw 1.0**, **defaultplex**, **enhanced**, "Helvetica" and 14pt. Default size of a PostScript plot is 10 inches wide and 7 inches high. The option **color** enables color, while **monochrome** prefers black and white drawing elements. Further, **monochrome** uses gray **palette** but it does not change color of objects specified with an explicit **colorspec**. **dashlength** or **dl** scales the length of dashed-line segments by <DL>, which is a floating-point number greater than zero. **linewidth** or **lw** scales all linewidths by <LW>.

By default the generated PostScript code uses language features that were introduced in PostScript Level 2, notably filters and pattern-fill of irregular objects such as filledcurves. PostScript Level 2 features are conditionally protected so that PostScript Level 1 interpreters do not issue errors but, rather, display a message or a PostScript Level 1 approximation. The **level1** option substitutes PostScript Level 1 approximations of these features and uses no PostScript Level 2 code. This may be required by some old printers and old versions of Adobe Illustrator. The flag **level1** can be toggled later by editing a single line in the PostScript output file to force PostScript Level 1 interpretation. In the case of files containing level 2 code, the above features will not appear or will be replaced by a note when this flag is set or when the interpreting program does not indicate that it understands level 2 PostScript or higher. The flag **level3** enables PNG encoding for bitmapped images, which can reduce the output size considerably.

rounded sets line caps and line joins to be rounded; **butt** is the default, butt caps and mitered joins.

clip tells PostScript to clip all output to the bounding box; **noclip** is the default.

palfuncparam controls how **set palette functions** are encoded as gradients in the output. Analytic color component functions (set via **set palette functions**) are encoded as linear interpolated gradients in the postscript output: The color component functions are sampled at <samples> points and all points are removed from this gradient which can be removed without changing the resulting colors by more than <maxdeviation>. For almost every useful palette you may safely leave the defaults of <samples>=2000 and <maxdeviation>=0.003 untouched.

The default size for postscript output is 10 inches x 7 inches. The default for eps output is 5 x 3.5 inches. The **size** option changes this to whatever the user requests. By default the X and Y sizes are taken to be in inches, but other units are possibly (currently only cm). The BoundingBox of the plot is correctly adjusted to contain the resized image. Screen coordinates always run from 0.0 to 1.0 along the full length of the plot edges as specified by the **size** option. NB: **this is a change from the previously recommended method of using the set size command prior to setting the terminal type**. The old method left the BoundingBox unchanged and screen coordinates did not correspond to the actual limits of the plot.

Fonts listed by **fontfile** or **fontfile add** encapsulate the font definitions of the listed font from a postscript Type 1 or TrueType font file directly into the gnuplot output postscript file. Thus, the enclosed font can be used in labels, titles, etc. See the section **postscript fontfile (p. 278)** for more details. With **fontfile delete**, a fontfile is deleted from the list of embedded files. **nofontfiles** cleans the list of embedded fonts.

Examples:

```
set terminal postscript default       # old postscript
set terminal postscript enhanced      # old enhpost
set terminal postscript landscape 22  # old psbig
set terminal postscript eps 14        # old epsf1
set terminal postscript eps 22        # old epsf2
set size 0.7,1.4; set term post portrait color "Times-Roman" 14
set term post "VAGRoundedBT_Regular" 14 fontfile "bvrr8a.pfa"
```

Linewidths and pointsizes may be changed with **set style line**.

The **postscript** driver supports about 70 distinct pointtypes, selectable through the **pointtype** option on **plot** and **set style line**.

Several possibly useful files about **gnuplot**'s PostScript are included in the /docs/psdoc subdirectory of the **gnuplot** distribution and at the distribution sites. These are "ps_symbols.gpi" (a **gnuplot** command file that, when executed, creates the file "ps_symbols.ps" which shows all the symbols available through the **postscript** terminal), "ps_guide.ps" (a PostScript file that contains a summary of the enhanced syntax and a page showing what the octal codes produce with text and symbol fonts), "ps_file.doc" (a text file that contains a discussion of the organization of a PostScript file written by **gnuplot**), and "ps_fontfile_doc.tex" (a LaTeX file which contains a short documentation concerning the encapsulation of LaTeX fonts with a glyph table of the math fonts).

A PostScript file is editable, so once **gnuplot** has created one, you are free to modify it to your heart's desire. See the **editing postscript (p. 278)** section for some hints.

Editing postscript

The PostScript language is a very complex language — far too complex to describe in any detail in this document. Nevertheless there are some things in a PostScript file written by **gnuplot** that can be changed without risk of introducing fatal errors into the file.

For example, the PostScript statement "/Color true def" (written into the file in response to the command **set terminal postscript color**), may be altered in an obvious way to generate a black-and-white version of a plot. Similarly line colors, text colors, line weights and symbol sizes can also be altered in straight-forward ways. Text (titles and labels) can be edited to correct misspellings or to change fonts. Anything can be repositioned, and of course anything can be added or deleted, but modifications such as these may require deeper knowledge of the PostScript language.

The organization of a PostScript file written by **gnuplot** is discussed in the text file "ps_file.doc" in the docs/ps subdirectory of the gnuplot source distribution.

Postscript fontfile

```
set term postscript ... fontfile {add|delete} <filename>
```

The **fontfile** or **fontfile add** option takes one file name as argument and encapsulates this file into the postscript output in order to make this font available for text elements (labels, tic marks, titles, etc.). The **fontfile delete** option also takes one file name as argument. It deletes this file name from the list of encapsulated files.

The postscript terminal understands some font file formats: Type 1 fonts in ASCII file format (extension ".pfa"), Type 1 fonts in binary file format (extension ".pfb"), and TrueType fonts (extension ".ttf"). pfa files are understood directly, pfb and ttf files are converted on the fly if appropriate conversion tools are installed (see below). You have to specify the full filename including the extension. Each **fontfile** option takes exact one font file name. This option can be used multiple times in order to include more than one font file.

The search order used to find font files is (1) absolute pathname or current working directory (2) any of the directories specified by **set loadpath** (3) the directory specified by **set fontpath** (4) the directory given in environmental variable GNUPLOT_FONTPATH. NB: This is a CHANGE from earlier versions of gnuplot.

For using the encapsulated font file you have to specify the font name (which normally is not the same as the file name). When embedding a font file by using the **fontfile** option in interactive mode, the font name is printed on the screen. E.g.

```
Font file 'p052004l.pfb' contains the font 'URWPalladioL-Bold'. Location:
/usr/lib/X11/fonts/URW/p052004l.pfb
```

When using pfa or pfb fonts, you can also find it out by looking into the font file. There is a line similar to "/FontName /URWPalladioL-Bold def". The middle string without the slash is the fontname, here

"URWPalladioL-Bold". For TrueType fonts, this is not so easy since the font name is stored in a binary format. In addition, they often have spaces in the font names which is not supported by Type 1 fonts (in which a TrueType is converted on the fly). The font names are changed in order to eliminate the spaces in the fontnames. The easiest way to find out which font name is generated for use with gnuplot, start gnuplot in interactive mode and type in "set terminal postscript fontfile '<filename.ttf>'".

For converting font files (either ttf or pfb) to pfa format, the conversion tool has to read the font from a file and write it to standard output. If the output cannot be written to standard output, on-the-fly conversion is not possible.

For pfb files "pfbtops" is a tool which can do this. If this program is installed on your system the on the fly conversion should work. Just try to encapsulate a pfb file. If the compiled in program call does not work correctly you can specify how this program is called by defining the environment variable GNUPLOT_PFBTOPFA e.g. to "pfbtops %s". The **%s** will be replaced by the font file name and thus has to exist in the string.

If you don't want to do the conversion on the fly but get a pfa file of the font you can use the tool "pfb2pfa" which is written in simple c and should compile with any c compiler. It is available from many ftp servers, e.g. `ftp://ftp.dante.de/tex-archive/fonts/utilities/ps2mf/`

In fact, "pfbtopfa" and "pfb2ps" do the same job. "pfbtopfa" puts the resulting pfa code into a file, whereas "pfbtops" writes it to standard output.

TrueType fonts are converted into Type 1 pfa format, e.g. by using the tool "ttf2pt1" which is available from `http://ttf2pt1.sourceforge.net/`

If the builtin conversion does not work, the conversion command can be changed by the environment variable GNUPLOT_TTFTOPFA. For usage with ttf2pt1 it may be set to "ttf2pt1 -a -e -W 0 %s - ". Here again, **%s** stands for the file name.

For special purposes you also can use a pipe (if available for your operating system). Therefore you start the file name definition with the character "<" and append a program call. This program has to write pfa data to standard output. Thus, a pfa file may be accessed by **set fontfile "< cat garamond.pfa"**.

For example, including Type 1 font files can be used for including the postscript output in LaTeX documents. The "european computer modern" font (which is a variant of the "computer modern" font) is available in pfb format from any CTAN server, e.g. `ftp://ftp.dante.de/tex-archive/fonts/ps-type1/cm-super/`

For example, the file "sfrm1000.pfb" contains the normal upright fonts with serifs in the design size 10pt (font name "SFRM1000"). The computer modern fonts, which are still necessary for mathematics, are available from `ftp://ftp.dante.de/tex-archive/fonts/cm/ps-type1/bluesky`

With these you can use any character available in TeX. However, the computer modern fonts have a strange encoding. (This is why you should not use cmr10.pfb for text, but sfrm1000.pfb instead.) The usage of TeX fonts is shown in one of the demos. The file "ps_fontfile_doc.tex" in the /docs/psdoc subdirectory of the **gnuplot** source distribution contains a table with glyphs of the TeX mathfonts.

If the font "CMEX10" is embedded (file "cmex10.pfb") gnuplot defines the additional font "CMEX10-Baseline". It is shifted vertically in order to fit better to the other glyphs (CMEX10 has its baseline at the top of the symbols).

Postscript prologue

Each PostScript output file includes a %%Prolog section and possibly some additional user-defined sections containing, for example, character encodings. These sections are copied from a set of PostScript prologue files that are either compiled into the gnuplot executable or stored elsewhere on your computer. A default directory where these files live is set at the time gnuplot is built. However, you can override this default either by using the gnuplot command **set psdir** or by defining an environment variable GNUPLOT_PS_DIR. See **set psdir (p. 193)**.

Postscript adobeglyphnames

This setting is only relevant to PostScript output with UTF-8 encoding. It controls the names used to describe characters with Unicode entry points higher than 0x00FF. That is, all characters outside of the Latin1 set. In general unicode characters do not have a unique name; they have only a unicode identification code. However, Adobe have a recommended scheme for assigning names to certain ranges of characters (extended Latin, Greek, etc). Some fonts use this scheme, others do not. By default, gnuplot will use the Adobe glyph names. E.g. the lower case Greek letter alpha will be called /alpha. If you specific **noadobeglyphnames** then instead gnuplot will use /uni03B1 to describe this character. If you get this setting wrong, the character may not be found even if it is present in the font. It is probably always correct to use the default for Adobe fonts, but for other fonts you may have to try both settings. See also **fontfile** **(p. 278)**.

Pslatex and pstex

The **pslatex** driver generates output for further processing by LaTeX, while the **pstex** driver generates output for further processing by TeX. **pslatex** uses \specials understandable by dvips and xdvi. Figures generated by **pstex** can be included in any plain-based format (including LaTeX).

Syntax:

```
set terminal [pslatex | pstex] {default}
set terminal [pslatex | pstex]
                        {rotate | norotate}
                        {oldstyle | newstyle}
                        {auxfile | noauxfile}
                        {level1 | leveldefault | level3}
                        {color | colour | monochrome}
                        {background <rgbcolor> | nobackground}
                        {dashlength | dl <DL>}
                        {linewidth | lw <LW>} {pointscale | ps <PS>}
                        {rounded | butt}
                        {clip | noclip}
                        {palfuncparam <samples>{,<maxdeviation>}}
                        {size <XX>{unit},<YY>{unit}}
                        {<font_size>}
```

If you see the error message

```
"Can't find PostScript prologue file ... "
```

Please see and follow the instructions in **postscript prologue** **(p. 279)**.

The option **color** enables color, while **monochrome** prefers black and white drawing elements. Further, **monochrome** uses gray **palette** but it does not change color of objects specified with an explicit **colorspec**. **dashlength** or **dl** scales the length of dashed-line segments by <DL>, which is a floating-point number greater than zero. **linewidth** or **lw** scales all linewidths by <LW>.

By default the generated PostScript code uses language features that were introduced in PostScript Level 2, notably filters and pattern-fill of irregular objects such as filledcurves. PostScript Level 2 features are conditionally protected so that PostScript Level 1 interpreters do not issue errors but, rather, display a message or a PostScript Level 1 approximation. The **level1** option substitutes PostScript Level 1 approximations of these features and uses no PostScript Level 2 code. This may be required by some old printers and old versions of Adobe Illustrator. The flag **level1** can be toggled later by editing a single line in the PostScript output file to force PostScript Level 1 interpretation. In the case of files containing level 2 code, the above features will not appear or will be replaced by a note when this flag is set or when the interpreting program does not indicate that it understands level 2 PostScript or higher. The flag **level3** enables PNG encoding for bitmapped images, which can reduce the output size considerably.

rounded sets line caps and line joins to be rounded; **butt** is the default, butt caps and mitered joins.

clip tells PostScript to clip all output to the bounding box; **noclip** is the default.

palfuncparam controls how **set palette functions** are encoded as gradients in the output. Analytic color component functions (set via **set palette functions**) are encoded as linear interpolated gradients in the postscript output: The color component functions are sampled at <samples> points and all points are removed from this gradient which can be removed without changing the resulting colors by more than <maxdeviation>. For almost every useful palette you may safely leave the defaults of <samples>=2000 and <maxdeviation>=0.003 untouched.

The default size for postscript output is 10 inches x 7 inches. The default for eps output is 5 x 3.5 inches. The **size** option changes this to whatever the user requests. By default the X and Y sizes are taken to be in inches, but other units are possibly (currently only cm). The BoundingBox of the plot is correctly adjusted to contain the resized image. Screen coordinates always run from 0.0 to 1.0 along the full length of the plot edges as specified by the **size** option. NB: **this is a change from the previously recommended method of using the set size command prior to setting the terminal type**. The old method left the BoundingBox unchanged and screen coordinates did not correspond to the actual limits of the plot.

if **rotate** is specified, the y-axis label is rotated. <font_size> is the size (in pts) of the desired font.

If **auxfile** is specified, it directs the driver to put the PostScript commands into an auxiliary file instead of directly into the LaTeX file. This is useful if your pictures are large enough that dvips cannot handle them. The name of the auxiliary PostScript file is derived from the name of the TeX file given on the **set output** command; it is determined by replacing the trailing **.tex** (actually just the final extent in the file name) with **.ps** in the output file name, or, if the TeX file has no extension, **.ps** is appended. The **.ps** is included into the **.tex** file by a \special{psfile=...} command. Remember to close the **output file** before next plot unless in **multiplot** mode.

Gnuplot versions prior to version 4.2 generated plots of the size 5 x 3 inches using the ps(la)tex terminal while the current version generates 5 x 3.5 inches to be consistent with the postscript eps terminal. In addition, the character width is now estimated to be 60% of the font size while the old epslatex terminal used 50%. To reach the old format specify the option **oldstyle**.

The pslatex driver offers a special way of controlling text positioning: (a) 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. (b) 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.

The options not described here are identical to the **Postscript terminal**. Look there if you want to know what they do.

Examples:
```
set term pslatex monochrome rotate       # set to defaults
```

To write the PostScript commands into the file "foo.ps":
```
set term pslatex auxfile
set output "foo.tex"; plot ...; set output
```

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}}'
```

Linewidths and pointsizes may be changed with **set style line**.

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 at http://tug.org/PSTricks/.

This driver definitely does not come close to using the full capability of the PSTricks package.

Syntax:

```
set terminal pstricks
               {unit | size <XX>{unit},<YY>{unit}}
               {standalone | input}
               {blacktext | colortext | colourtext}
               {linewidth <lw>} {rounded | butt}
               {pointscale <ps>}
               {psarrows | gparrows}
               {background <rgbcolor>}
```

The **unit** option produces a plot with internal dimensions 1x1. The default is a plot of **size 5in,3in**.

standalone produces a LaTeX file with possibly multiple plots, ready to be compiled. The default is **input** to produce a TeX file which can be included.

blacktext forces all text to be written in black. **colortext** enables colored text. The default is **blacktext**.

rounded sets line caps and line joins to be rounded. **butt** sets butt caps and mitered joins and is the default.

linewidth and **pointscale** scale the width of lines and the size of point symbols, respectively.

psarrows draws **arrow**s using PSTricks commands which are shorter but do not offer all options. **gparrows** selects drawing arrows using gnuplot's own routine for full functionality instead.

The old **hacktext** option has been replaced by the new default format (%h), see **format specifiers (p. 149)**.

Transparency requires support by Ghostscript or conversion to PDF.

Qms

The **qms** terminal driver supports the QMS/QUIC Laser printer, the Talaris 1200 and others. It has no options.

Qt

The **qt** terminal device generates output in a separate window with the Qt library. Syntax:

```
set term qt {<n>}
            {size <width>,<height>}
            {position <x>,<y>}
            {title "title"}
            {font <font>} {{no}enhanced}
            {linewidth <lw>} {dashlength <dl>}
            {{no}persist} {{no}raise} {{no}ctrl}
            {close}
            {widget <id>}
```

Multiple plot windows are supported: **set terminal qt <n>** directs the output to plot window number n.

The default window title is based on the window number. This title can also be specified with the keyword "title".

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, by choosing **close** from a window manager menu, or with **set term qt <n> close**.

The size of the plot area is given in pixels, it defaults to 640x480. In addition to that, the actual size of the window also includes the space reserved for the toolbar and the status bar. When you resize a window, the plot is immediately scaled to fit in the new size of the window. The **qt** terminal scales the whole plot, including fonts and linewidths, and keeps its global aspect ratio constant. If you type **replot**, click the **replot** icon in the terminal toolbar or type a new **plot** command, the new plot will completely fit in the window and the font size and the linewidths will be reset to their defaults.

The position option can be used to set the position of the plot window. The position option only applies to the first plot after the **set term** command.

The active plot window (the one selected by **set term qt <n>**) is interactive. Its behaviour is shared with other terminal types. See **mouse (p. 169)** for details. It also has some extra icons, which are supposed to be self-explanatory.

This terminal supports an enhanced text mode, which allows font and other formatting commands (subscripts, superscripts, etc.) to be embedded in labels and other text strings. The enhanced text mode syntax is shared with other gnuplot terminal types. See **enhanced (p. 34)** for more details.

<font> is in the format "FontFace,FontSize", i.e. the face and the size comma-separated in a single string. FontFace is a usual font face name, such as 'Arial'. If you do not provide FontFace, the qt terminal will use 'Sans'. FontSize is the font size, in points. If you do not provide it, the qt terminal will use a size of 9 points.

```
For example :
   set term qt font "Arial,12"
   set term qt font "Arial" # to change the font face only
   set term qt font ",12" # to change the font size only
   set term qt font "" # to reset the font name and size
```

The dashlength affects only custom dash patterns, not Qt's built-in set.

To obtain the best output possible, the rendering involves three mechanisms : antialiasing, oversampling and hinting. Oversampling combined with antialiasing provides subpixel accuracy, so that gnuplot can draw a line from non-integer coordinates. This avoids wobbling effects on diagonal lines ('plot x' for example). Hinting avoids the blur on horizontal and vertical lines caused by oversampling. The terminal will snap these lines to integer coordinates so that a one-pixel-wide line will actually be drawn on one and only one pixel.

By default, the window is raised to the top of your desktop when a plot is drawn. This can be controlled with the keyword "raise". The keyword "persist" will prevent gnuplot from exiting before you explicitly close all the plot windows.

The <space> key raises the gnuplot console window [MS Windows only]. The 'q' key closes the plot window. These hot keys can be changed to ctrl-space and ctrl-q using the terminal option keyword "{no}ctrl". However the preferred way to select ctrl-q rather than 'q' is to use the toggle in the tools widget of the plot window.

The gnuplot outboard driver, gnuplot_qt, is searched in a default place chosen when the program is compiled. You can override that by defining the environment variable GNUPLOT_DRIVER_DIR.

Regis

Note: legacy terminal. 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}
```

Sixelgd

Syntax:

```
set terminal sixelgd
       {{no}enhanced} {{no}truecolor}
       {{no}transparent} {rounded|butt}
       {linewidth <lw>} {dashlength <dl>}
       {tiny | small | medium | large | giant}
       {font "<face> {,<pointsize>}"} {fontscale <scale>}
       {size <x>,<y>} {{no}crop} {animate}
       {background <rgb_color>}
```

The **sixel** output format was originally used by DEC terminals and printers. This driver produces a sixel output stream by converting a PNG image created internally using the gd library. The sixel output stream can be viewed in the terminal as it is created or it can be written to a file so that it can be replayed later by echoing the file to the terminal.

The **animate** option resets the cursor position to the terminal top left at the start of every plot so that successive plots overwrite the same area on the screen rather than having earlier plots scroll off the top. This may be desirable in order to create an in-place animation.

transparent instructs the driver to make the background color transparent. Default is **notransparent**.

The **linewidth** and **dashlength** options are scaling factors that affect all lines drawn, i.e. they are multiplied by values requested in various drawing commands.

By default the sixel output uses 16 indexed colors. The **truecolor** option instead creates a TrueColor png image that is mapped down onto 256 colors in the output sixel image. Transparent fill styles require the **truecolor** option. See **fillstyle (p. 199)**. A **transparent** background is possible in either indexed or TrueColor images.

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**.

The details of font selection are complicated. For more information please see **fonts (p. 45)**.

The output plot size <x,y> is given in pixels — it defaults to 640x480. Please see additional information under **canvas (p. 31)** and **set size (p. 195)**. Blank space at the edges of the finished plot may be trimmed using the **crop** option, resulting in a smaller final image size. Default is **nocrop**. The terminal has been successfully tested with the xterm, mlterm and mintty terminals. The later two support the **truecolor** mode using 256 sixel colors out of box. Distributed copies of xterm may or may not have been configured to support sixel graphics and may be limited to 16 colors.

Svg

This terminal produces files in the W3C Scalable Vector Graphics format.

Syntax:

```
set terminal svg {size <x>,<y> {|fixed|dynamic}}
                 {mouse} {standalone | jsdir <dirname>}
                 {name <plotname>}
                 {font "<fontname>{,<fontsize>}"} {{no}enhanced}
                 {fontscale <multiplier>}
                 {rounded|butt|square} {solid|dashed} {linewidth <lw>}
                 {background <rgb_color>}
```

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). SVG viewing programs may substitute other fonts when the file is displayed.

The enhanced text mode syntax is shared with other gnuplot terminal types. See **enhanced (p. 34)** for more details.

The **mouse** option tells gnuplot to add support for mouse tracking and for toggling individual plots on/off by clicking on the corresponding key entry. By default this is done by including a link that points to a script in a local directory, usually /usr/local/share/gnuplot/<version>/js. You can change this by using the **jsdir** option to specify either a different local directory or a general URL. The latter is usually appropriate if you are embedding the svg into a web page. Alternatively, the **standalone** option embeds the mousing code in the svg document itself rather than linking to an external resource.

When an SVG file will be used in conjunction with external files, e.g. if it is referenced by javascript code in a web page or parent document, then a unique name is required to avoid potential conflicting references to other SVG plots. Use the **name** option to ensure uniqueness.

Svga

Legacy terminal. The **svga** terminal driver supports PCs with SVGA graphics. It is typically only compiled with DJGPP and uses the GRX graphics library. There is also a variant for Windows 32bit, which is mainly used for debugging. The underlying library also supports X11, Linux console and SDL, but these targets are currently not supported.

Syntax:

```
set terminal svga {font "<fontname>"}
                  {{no}enhanced}
                  {background <rgb color>}
                  {linewidth|lw <lw>}
                  {pointscale|ps <scale>}
                  {fontscale|fs <scale>}
```

Enhanced text support can be activated using the **enhanced** option, see **enhanced text (p. 34)**. Note that changing the font size in enhanced text is currently not supported. Hence, super- and subscripts will have the same size.

The **linewidth** parameter scales the width of lines. The **pointscale** parameter sets the scale factor for point symbols. You can use **fontscale** to scale the bitmap font. This might be useful if you have a hi-res display. Note that integer factors give best results.

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. The following are present only if selected when gnuplot is built: **kc-tek40xx** supports MS-DOS Kermit Tek4010 terminal emulators in color; **km-tek40xx** supports them in monochrome. **selanar** supports Selanar graphics. **bitgraph** supports BBN Bitgraph terminals. None have any options.

Tek410x

The **tek410x** terminal driver supports the 410x and 420x family of Tektronix terminals. It has no options.

Texdraw

The **texdraw** terminal driver supports the (La)TeX texdraw environment. It is intended for use with the texdraw package, see https://www.ctan.org/tex-archive/graphics/texdraw/ .

```
set terminal texdraw
               {size <XX>{unit},<YY>{unit}}
               {standalone | input}
               {blacktext | colortext | colourtext}
               {linewidth <lw>} {rounded | butt}
               {pointscale <ps>}
               {psarrows | gparrows} {texpoints | gppoints}
               {background <rgbcolor>}
```

Note: Graphics are in grayscale only. Text is always black. Boxes and polygons are filled using solid gray levels only. Patterns are not available.

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. Other point types use symbols from the amssymb package. For compatibility with plain TeX you need to specify the **gppoints** option.

standalone produces a LaTeX file with possibly multiple plots, ready to be compiled. The default is **input** to produce a TeX file which can be included.

blacktext forces all text to be written in black. **colortext** enables "colored" text. The default is **blacktext** and "color" means grayscale really.

rounded sets line caps and line joins to be rounded; **butt** sets butt caps and mitered joins and is the default.

linewidth and **pointscale** scale the width of lines and the size of point symbols, respectively. **pointscale**only applies to **gppoints**.

psarrows draws **arrow**s using TeXdraw commands which are shorter but do not offer all options. **gparrows** selects drawing drawing arrows using gnuplot's own routine for full functionality instead. Similarly, **texpoints**, and **gppoints** select LaTeX symbols or gnuplot's point drawing routines.

Tgif

Legacy terminal (present only if gnuplot was configured –with-tgif). Tgif is/was an Xlib based interactive 2-D vector graphics drawing tool also capable of importing and marking up bitmap images.

The **tgif** driver supports a choice of font and font size 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**, **linewidth 1.0**, **dashed**, **"Helvetica,18"**.

The **solid** option is usually preferred 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 **set multiplot (p. 171)** 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 **set size**).

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
```

Tikz

This driver creates output for use with the TikZ package of graphics macros in TeX. It is currently implemented via an external lua script, and **set term tikz** is a short form of the command **set term lua tikz**. See **term lua (p. 263)** for more information. Use the command **set term tikz help** to print terminal options.

Tkcanvas

This terminal driver generates Tk canvas widget commands in one of the following scripting languages: Tcl (default), Perl, Python, Ruby, or REXX.

Syntax:

```
set terminal tkcanvas {tcl | perl | perltkx | python | ruby | rexx}
                      {standalone | input}
                      {interactive}
                      {rounded | butt}
                      {nobackground | background <rgb color>}
                      {{no}rottext}
```

```
                           {size <width>,<height>}
                           {{no}enhanced}
                           {externalimages | pixels}
```

Execute the following sequence of Tcl/Tk commands to display the result:

```
package require Tk
# the following two lines are only required to support external images
package require img::png
source resize.tcl
source plot.tcl
canvas .c -width 800 -height 600
pack .c
gnuplot .c
```

Or, for Perl/Tk use a program like this:

```
use Tk;
my $top = MainWindow->new;
my $c = $top->Canvas(-width => 800, -height => 600)->pack;
my $gnuplot = do "plot.pl";
$gnuplot->($c);
MainLoop;
```

Or, for Perl/Tkx use a program like this:

```
use Tkx;
my $top = Tkx::widget->new(".");
my $c = $top->new_tk__canvas(-width => 800, -height => 600);
$c->g_pack;
my $gnuplot = do "plot.pl";
$gnuplot->($c);
Tkx::MainLoop();
```

Or, for Python/Tkinter use a program like this:

```
from tkinter import *
from tkinter import font
root = Tk()
c = Canvas(root, width=800, height=600)
c.pack()
exec(open('plot.py').read())
gnuplot(c)
root.mainloop()
```

Or, for Ruby/Tk use a program like this:

```
require 'tk'
root = TkRoot.new { title 'Ruby/Tk' }
c = TkCanvas.new(root, 'width'=>800, 'height'=>600) { pack  { } }
load('plot.rb')
gnuplot(c)
Tk.mainloop
```

Or, for Rexx/Tk use a program like this:

```
/**/
call RxFuncAdd 'TkLoadFuncs', 'rexxtk', 'TkLoadFuncs'
```

```
call TkLoadFuncs
cv = TkCanvas('.c', '-width', 800, '-height', 600)
call TkPack cv
call 'plot.rex' cv
do forever
    cmd = TkWait()
    if cmd = 'AWinClose' then leave
    interpret 'call' cmd
end
```

The code generated by **gnuplot** (in the above examples, this code is written to "plot.<ext>") contains the following procedures:

gnuplot(canvas)

```
takes the name of a canvas as its argument.
When called, it clears the canvas, finds the size of the canvas and
draws the plot in it, scaled to fit.
```

gnuplot_plotarea()

```
returns a list containing the borders of the plotting area
(xleft, xright, ytop, ybot) in canvas screen coordinates.    It works only for 2-dimensional plotting
```

gnuplot_axisranges()

```
returns the ranges of the two axes in plot coordinates
(x1min, x1max, y1min, y1max, x2min, x2max, y2min, y2max).
It works only for 2-dimensional plotting (`plot`).
```

You can create self-contained, minimal scripts using the **standalone** option. The default is **input** which creates scripts which have to be source'd (or loaded or called or whatever the adequate term is for the language selected).

If the **interactive** option is specified, mouse clicking on a line segment will print the coordinates of its midpoint to stdout. The user can supersede this behavior by supplying a procedure user_gnuplot_coordinates which takes the following arguments:

```
win id x1s y1s x2s y2s x1e y1e x2e y2e x1m y1m x2m y2m,
```

i.e. 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.

By default the canvas is **transparent**, but an explicit background color can be set with the **background** option.

rounded sets line caps and line joins to be rounded; **butt** is the default: butt caps and mitered joins.

Text at arbitrary angles can be activated with the **rottext** option, which requires Tcl/Tk 8.6 or later. The default is **norottext**.

The **size** option tries to optimize the tic and font sizes for the given canvas size. By default an output size of 800 x 600 pixels is assumed.

enhanced selects **enhanced text** processing (default), but is currently only available for Tcl.

The **pixels** (default) option selects the failsafe pixel-by-pixel image handler, see also **image pixels (p. 74)**. The **externalimages** option saves images as external png images, which are later loaded and scaled by the tkcanvas code. This option is only available for Tcl and display may be slow in some situations since the Tk image handler does not provide arbitrary scaling. Scripts need to source the provided rescale.tcl.

Interactive mode is not yet implemented for Python/Tk and Rexx/Tk. Interactive mode for Ruby/Tk does not yet support user_gnuplot_coordinates.

Tpic

Note: Legacy terminal (not built by default). The latex, emtex, eepic, and tpic terminals in older versions of gnuplot provided minimal support for graphics styles beyond simple lines and points. They have been directly superseded by the **pict2e** terminal. For more capable TeX/LaTeX compatible terminal types see **cairolatex (p. 240)**, **context (p. 246)**, **epslatex (p. 252)**, **mp (p. 267)**, **pstricks (p. 282)**, and **tikz (p. 287)**.

The **tpic** terminal driver supports the LaTeX picture environment with tpic \specials. Options are the point size, line width, and dot-dash interval.

Syntax:

```
set terminal tpic <pointsize> <linewidth> <interval>
```

where **pointsize** 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}}'
```

VWS

Note: legacy terminal. 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.

Windows

The **windows** terminal is a fast interactive terminal driver that uses the Windows GDI to draw and write text. The cross-platform **terminal wxt** and **terminal qt** are also supported on Windows.

Syntax:

```
set terminal windows {<n>}
                     {color | monochrome}
                     {solid | dashed}
                     {rounded | butt}
                     {enhanced | noenhanced}
                     {font <fontspec>}
                     {fontscale <scale>}
                     {linewidth <scale>}
                     {pointscale <scale>}
```

```
{background <rgb color>}
{title "Plot Window Title"}
{{size | wsize} <width>,<height>}
{position <x>,<y>}
{docked {layout <rows>,<cols>} | standalone}
{close}
```

Multiple plot windows are supported: **set terminal win <n>** directs the output to plot window number n.

color and **monochrome** select colored or mono output, **dashed** and **solid** select dashed or solid lines. Note that **color** defaults to **solid**, whereas **monochrome** defaults to **dashed**. **rounded** sets line caps and line joins to be rounded; **butt** is the default, butt caps and mitered joins. **enhanced** enables enhanced text mode features (subscripts, superscripts and mixed fonts, see **enhanced text (p. 34)** for more information). **<fontspec>** is in the format "<fontface>,<fontsize>", where "<fontface>" is the name of a valid Windows font, and <fontsize> is the size of the font in points and both components are optional. Note that in previous versions of gnuplot the **font** statement could be left out and <fontsize> could be given as a number without double quotes. This is no longer supported. **linewidth**, **fontscale**, **pointscale** can be used to scale the width of lines, the size of text, or the size of the point symbols. **title** changes the title of the graph window. **size** defines the width and height of the window's drawing area in pixels, **wsize** defines the actual size of the window itself and **position** defines the origin of the window i.e. the position of the top left corner on the screen (again in pixel). These options override any default settings from the **wgnuplot.ini** file.

docked embeds the graph window in the wgnuplot text window and the **size** and **position** options are ignored. Note that **docked** is not available for console-mode gnuplot. Setting this option changes the default for new windows. The initial default is **standalone**. The **layout** option allows to reserve a minimal number of columns and rows for graphs in docked mode. If there are more graphs than fit the given layout, additional rows will be added. Graphs are sorted by the numerical id, filling rows first.

Other options may be changed using the **graph-menu** or the initialization file **wgnuplot.ini**.

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.

The plot window remains open when the gnuplot terminal is changed with a **set term** command. The plot window can be closed with **set term windows close**.

gnuplot supports different methods to create printed output on Windows, see **windows printing (p. 292)**. The windows terminal supports data exchange with other programs via clipboard and EMF files, see **graph-menu (p. 291)**. You can also use the **terminal emf** to create EMF files.

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 or the toolbar:

Copy to Clipboard copies a bitmap and an enhanced metafile picture.

Save as EMF... allows the user to save the current graph window as enhanced metafile (EMF or EMF+).

Save as Bitmap... allows the user to save a copy of the graph as bitmap file.

Print... prints the graphics windows using a Windows printer driver and allows selection of the printer and scaling of the output. See also **windows printing (p. 292)**.

Bring to Top when checked raises the graph window to the top after every plot.

Color when checked enables color output. When unchecked it forces all grayscale output. This is e.g. useful to test appearance of monochrome printouts.

The **GDI backend** which uses the classic GDI API is deprecated and has been disabled in this version.

GDI+ backend draws to the screen using the GDI+ Windows API. It supports full antialiasing, oversampling, transparency and custom dash patterns. This was the default in versions 5.0 and 5.2.

Direct2D backend uses Direct2D & DirectWrite APIs to draw. It uses graphic card acceleration and is hence typically much faster. Since Direct2D can not create EMF data, saving and copying to clipboard of EMF data fall back to GDI+ while bitmap data is generated by D2d. This is the recommended and default backend since version 5.3.

Oversampling draws diagonal lines at fractional pixel positions to avoid "wobbling" effects. Vertical or horizontal lines are still snapped to integer pixel positions to avoid blurry lines.

Antialiasing enables smoothing of lines and edges. Note that this slows down drawing. **Antialiasing of polygons** is enabled by default but might slow down drawing with the GDI+ backend.

Fast rotation switches antialiasing temporarily off while rotating the graph with the mouse. This speeds up drawing considerably at the expense of an additional redraw after releasing the mouse button.

Background... sets the window background color.

Choose Font... selects the font used in the graphics window.

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 to the initialization file **wgnuplot.ini**.

(*) Note that this menu is only available by pressing the right mouse button with **unset mouse**.

Printing

In order of preference, graphs may be printed in the following ways:

1. Use the **gnuplot** command **set terminal** to select a printer and **set output** 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 **set output** 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.

Wrap long lines when selected lines longer than the current window width are wrapped.

Update wgnuplot.ini saves the current settings to the initialisation file **wgnuplot.ini**, which is located in the user's application data directory.

Wgnuplot.mnu

If the menu file **wgnuplot.mnu** is found in the same directory as **gnuplot**, 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 a file to open, with the title of the dialog
            terminated by [EOS], followed by a default filename terminated
            by [EOS] or {ENTER}.
[SAVE]      Get name of a file to save.  Parameters like [OPEN]
[DIRECTORY] Get name of a directory, with the title of the dialog
            terminated by [EOS] or {ENTER}
```

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

The Windows text window and the **windows** terminal will read some of their options from the [**WGNUPLOT**] section of **wgnuplot.ini**. This file is located in the user's application data directory. Here's a sample **wgnuplot.ini** file:

```
[WGNUPLOT]
TextOrigin=0 0
TextSize=640 150
TextFont=Consolas,9
TextWrap=1
TextLines=400
TextMaximized=0
SysColors=0
GraphOrigin=0 150
GraphSize=640 330
GraphFont=Tahoma,10
GraphColor=1
GraphToTop=1
GraphGDI+=1
GraphD2D=0
GraphGDI+Oversampling=1
GraphAntialiasing=1
```

```
GraphPolygonAA=1
GraphFastRotation=1
GraphBackground=255 255 255
DockVerticalTextFrac=350
DockHorizontalTextFrac=400
```

These settings apply to the wgnuplot text-window only. The **TextOrigin** and **TextSize** entries specify the location and size of the text window. If **TextMaximized** is non-zero, the window will be maximized.

The **TextFont** entry specifies the text window font and size.

The **TextWrap** entry selects wrapping of long text lines.

The **TextLines** entry specifies the number of (unwrapped) lines the internal buffer of the text window can hold. This value currently cannot be changed from within wgnuplot.

See **text-menu (p. 292)**.

DockVerticalTextFrac and **DockHorizontalTextFrac** set the fraction of the window reserved for the text window in permille of the vertical or horizontal layout.

The **GraphFont** entry specifies the font name and size in points.

See **graph-menu (p. 291)**.

Wxt

The **wxt** terminal device generates output in a separate window. The window is created by the wxWidgets library, where the 'wxt' comes from. The actual drawing is done via cairo, a 2D graphics library, and pango, a library for laying out and rendering text.

Syntax:

```
set term wxt {<n>}
             {size <width>,<height>} {position <x>,<y>}
             {background <rgb_color> | nobackground}
             {{no}enhanced}
             {font <font>} {fontscale <scale>}
             {title "title"}
             {linewidth <lw>} {butt|rounded|square}
             {dashlength <dl>}
             {{no}persist}
             {{no}raise}
             {{no}ctrl}
             {close}
```

Multiple plot windows are supported: **set terminal wxt <n>** directs the output to plot window number n.

The default window title is based on the window number. This title can also be specified with the keyword "title".

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, by choosing **close** from a window manager menu, or with **set term wxt <n> close**.

The size of the plot area is given in pixels, it defaults to 640x384. In addition to that, the actual size of the window also includes the space reserved for the toolbar and the status bar. When you resize a window, the plot is immediately scaled to fit in the new size of the window. Unlike other interactive terminals, the **wxt** terminal scales the whole plot, including fonts and linewidths, and keeps its global aspect ratio constant, leaving an empty space painted in gray. If you type **replot**, click the **replot** icon in the terminal toolbar or type a new **plot** command, the new plot will completely fit in the window and the font size and the linewidths will be reset to their defaults.

The position option can be used to set the position of the plot window. The position option only applies to the first plot after the **set term** command.

The active plot window (the one selected by **set term wxt <n>**) is interactive. Its behaviour is shared with other terminal types. See **mouse (p. 169)** for details. It also has some extra icons, which are supposed to be self-explanatory.

This terminal supports an enhanced text mode, which allows font and other formatting commands (subscripts, superscripts, etc.) to be embedded in labels and other text strings. The enhanced text mode syntax is shared with other gnuplot terminal types. See **enhanced (p. 34)** for more details.

<font> is in the format "FontFace,FontSize", i.e. the face and the size comma-separated in a single string. FontFace is a usual font face name, such as 'Arial'. If you do not provide FontFace, the wxt terminal will use 'Sans'. FontSize is the font size, in points. If you do not provide it, the wxt terminal will use a size of 10 points.

```
For example :
   set term wxt font "Arial,12"
   set term wxt font "Arial" # to change the font face only
   set term wxt font ",12" # to change the font size only
   set term wxt font "" # to reset the font name and size
```

The fonts are retrieved from the usual fonts subsystems. Under Windows, those fonts are to be found and configured in the entry "Fonts" of the control panel. Under UNIX, they are handled by "fontconfig".

Pango, the library used to layout the text, is based on utf-8. Thus, the wxt terminal has to convert from your encoding to utf-8. The default input encoding is based on your 'locale'. If you want to use another encoding, make sure gnuplot knows which one you are using. See **encoding (p. 146)** for more details.

Pango may give unexpected results with fonts that do not respect the unicode mapping. With the Symbol font, for example, the wxt terminal will use the map provided by http://www.unicode.org/ to translate character codes to unicode. Pango will do its best to find a font containing this character, looking for your Symbol font, or other fonts with a broad unicode coverage, like the DejaVu fonts. Note that "the Symbol font" is to be understood as the Adobe Symbol font, distributed with Acrobat Reader as "SY______.PFB". Alternatively, the OpenSymbol font, distributed with OpenOffice.org as "opens___.ttf", offers the same characters. Microsoft has distributed a Symbol font ("symbol.ttf"), but it has a different character set with several missing or moved mathematic characters. If you experience problems with your default setup (if the demo enhancedtext.dem is not displayed properly for example), you probably have to install one of the Adobe or OpenOffice Symbol fonts, and remove the Microsoft one. Other non-conform fonts, such as "wingdings" have been observed working.

The rendering of the plot can be altered with a dialog available from the toolbar. To obtain the best output possible, the rendering involves three mechanisms : antialiasing, oversampling and hinting. Antialiasing allows to display non-horizontal and non-vertical lines smoother. Oversampling combined with antialiasing provides subpixel accuracy, so that gnuplot can draw a line from non-integer coordinates. This avoids wobbling effects on diagonal lines ('plot x' for example). Hinting avoids the blur on horizontal and vertical lines caused by oversampling. The terminal will snap these lines to integer coordinates so that a one-pixel-wide line will actually be drawn on one and only one pixel.

By default, the window is raised to the top of your desktop when a plot is drawn. This can be controlled with the keyword "raise". The keyword "persist" will prevent gnuplot from exiting before you explicitly close all the plot windows. Finally, by default the key <space> raises the gnuplot console window, and 'q' closes the plot window. The keyword "ctrl" allows you to replace those bindings by <ctrl>+<space> and <ctrl>+'q', respectively. These three keywords (raise, persist and ctrl) can also be set and remembered between sessions through the configuration dialog.

X11

Syntax:

```
set terminal x11 {<n> | window "<string>"}
                 {title "<string>"}
                 {{no}enhanced} {font <fontspec>}
                 {linewidth LW}
                 {{no}persist} {{no}raise} {{no}ctrlq}
                 {{no}replotonresize}
                 {close}
                 {size XX,YY} {position XX,YY}
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 can connect to X windows previously created by an outside application via the option **window** followed by a string containing the X ID for the window in hexadecimal format. Gnuplot uses that external X window as a container since X does not allow for multiple clients selecting the ButtonPress event. In this way, gnuplot's mouse features work within the contained plot window.

```
set term x11 window "220001e"
```

The x11 terminal supports enhanced text mode (see **enhanced (p. 34)**), 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 **reset**, 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 **reset**, 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 **raise** 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. If the window does not get raised, see discussion in **raise (p. 126)**.

The option **replotonresize** (active by default) replots the data when the plot window is resized. Without this option, the even-aspect-ratio scaling may result in the plot filling only part of the window after resizing. With this option, gnuplot does a full replot on each resize event, resulting in better space utilization. This option is generally desirable, unless the potentially CPU-intensive replotting during resizing is a concern. Replots can be manually initiated with hotkey 'e' or the 'replot' command.

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 option can be used to set the size of the plot window. The size option will only apply to newly created windows.

The position option can be used to set the position of the plot window. The position option will only apply to newly created windows.

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 **raise**)

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 `set encoding`).
```

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. You may set a default encoding via the corresponding X11 resource. E.g.

```
gnuplot*encoding: iso8859-15
```

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 **set label**.

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 **raise** and **persist** can be overridden later by **set term x11 [no]raise [no]persist**):

'-mono'	forces monochrome rendering on color displays.
'-gray'	requests grayscale rendering on grayscale or color displays. (Grayscale displays receive monochrome rendering 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 displayed portion of the virtual root.
'-raise'	raises plot window after each plot.
'-noraise'	does not raise plot window after each plot.
'-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 **-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.

Color_resources

NB: THIS SECTION IS LARGELY IRRELEVANT IN GNUPLOT VERSION 5 The X11 terminal 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

NB: THIS SECTION IS LARGELY IRRELEVANT IN GNUPLOT VERSION 5 **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: 1
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. Color displays default to dashed:off

```
gnuplot*dashed: off
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

NB: THIS SECTION IS LARGELY IRRELEVANT IN GNUPLOT VERSION 5 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

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 output "|gnuplot_x11 -noevents"; set term xlib**. **xlib** takes the same set of options as **x11**.

Part V

Bugs

Please e-mail bug reports to the gnuplot-bugs mailing list or upload the report to the gnuplot web site on SourceForge. Please give complete information on the version of gnuplot you are using and, if possible, a test script that demonstrates the bug. See **seeking-assistance (p. 22)**.

Known limitations

It is not possible to use inline data (e.g. plot '-' ...) inside the curly brackets of a **do** or **while** loop.

X11 terminal: It is difficult to select UTF-8 fonts. Only one color palette at a time is active for any given x11 plot window. This means that multiplots whose constituent plots use different palettes will not display correctly in x11.

Qt terminal: 3D rotation of polygons and surfaces can be slow; this is strongly affected by the Qt rendering mode (see Qt documentation).

The **raise** and **lower** commands are unreliable.

External libraries

External library GD (used by PNG/JPEG/GIF/sixelgd terminals, pixmap): Versions of libgd through 2.0.33 had various bugs in mapping the characters of Adobe's Symbol font. Dot-dash patterns are not supported for these terminals.

External library PDFlib (used by PDF terminal, but not by pdfcairo): Gnuplot can be linked against libpdf versions 4, 5, or 6. However, these versions differ in their handling of piped I/O. Therefore gnuplot scripts using piped output to PDF may work only for some versions of PDFlib.

Internationalization (locale settings): Gnuplot uses the C runtime library routine setlocale() to control locale-specific formatting of input and output number, times, and date strings. The locales available, and the level of support for locale features such as "thousands' grouping separator", depend on the internationalization support provided by your individual machine.

External library libcerf versions 1.8 1.9 1.10 return incorrect results for the voigt function. Do not use.

Part VI

Index

Index