gnuplot

An Interactive Plotting Program

Thomas Williams & Colin Kelley

Version 3.7 organized by: David Denholm

Major contributors (alphabetic order):
Hans-Bernhard Broeker
John Campbell
Robert Cunningham
David Denholm
Gershon Elber
Roger Fearick
Carsten Grammes
Lucas Hart
Lars Hecking
Thomas Koenig
David Kotz
Ed Kubaitis
Russell Lang
Alexander Lehmann
Alexander Mai
Carsten Steger
Tom Tkacik
Jos Van der Woude
James R. Van Zandt
Alex Woo

Mailing list for comments: `info-gnuplot@dartmouth.edu`
Mailing list for bug reports: `bug-gnuplot@dartmouth.edu`

This manual was prepared by Dick Crawford.
3 December 1998

Contents

Part I

Gnuplot

1 Copyright


```
AUTHORS

Original Software:
   Thomas Williams,  Colin Kelley.

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

Gnuplot 3.0 additions:
   Gershon Elber and many others.
```

2 Introduction

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

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

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

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

Many **gnuplot** commands have multiple options. These options must appear in the proper order, although unwanted ones may be omitted in most cases. Thus if the entire command is "command a b c", then "command a c" will probably work, but "command c a" will fail.

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 **comment**). 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 on-line help on any topic, type **help** followed by the name of the topic or just **help** or **?** to get a menu of available topics.

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

3 Seeking-assistance

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

`comp.graphics.apps.gnuplot`

is identical to the mailing list (they both carry the same set of messages). We prefer that you read the messages through the newsgroup rather than subscribing to the mailing list. Administrative requests should be sent to

`majordomo@dartmouth.edu`

Send a message with the body (not the subject) consisting of the single word "help" (without the quotes) for more details.

The address for mailing to list members is:

`info-gnuplot@dartmouth.edu`

Bug reports and code contributions should be mailed to:

`bug-gnuplot@dartmouth.edu`

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

`info-gnuplot-beta@dartmouth.edu`

There is also a World Wide Web page with up-to-date information, including known bugs:

`http://www.cs.dartmouth.edu/gnuplot_info.html`

Before seeking help, please check the FAQ (Frequently Asked Questions) list. If you do not have a copy of the FAQ, you may request a copy by email from the Majordomo address above, ftp a copy from

```
ftp://ftp.ucc.ie/pub/gnuplot/faq,
ftp://ftp.gnuplot.vt.edu/pub/gnuplot/faq,
```

or see the WWW **gnuplot** page.

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

4 What's New in version 3.7

Gnuplot version 3.7 contains many new features. This section gives a partial list and links to the new items in no particular order.

1. **fit f(x) 'file' via** uses the Marquardt-Levenberg method to fit data. (This is only slightly different from the **gnufit** patch available for 3.5.)

2. Greatly expanded **using** command. See **plot using**.

3. **set timefmt** allows for the use of dates as input and output for time series plots. See **Time/Date data** and timedat.dem.

4. Multiline labels and font selection in some drivers.

5. Minor (unlabeled) tics. See **set mxtics**.

6. **key** options for moving the key box in the page (and even outside of the plot), putting a title on it and a box around it, and more. See **set key**.

7. Multiplots on a single logical page with **set multiplot**.

8. Enhanced **postscript** driver with super/subscripts and font changes. (This was a separate driver (**enhpost**) that was available as a patch for 3.5.)

9. Second axes: use the top and right axes independently of the bottom and left, both for plotting and labels. See **plot**.

10. Special datafile names **'-'** and `""`. See **plot special-filenames**.

11. Additional coordinate systems for labels and arrows. See **coordinates**.

12. **set size** can try to plot with a specified aspect ratio.

13. **set missing** now treats missing data correctly.

14. The **call** command: **load** with arguments.

15. More flexible **range** commands with **reverse**, **writeback** and 'restore' keywords.

16. **set encoding** for multi-lingual encoding.

17. New **x11** driver with persistent and multiple windows.

18. New plotting styles: **xerrorbars**, **histeps**, **financebars** and more. See **set style**.

19. New tic label formats, including **"%l %L"** which uses the mantissa and exponents to a given base for labels. See **set format**.

20. New drivers, including **cgm** for inclusion into MS-Office applications and **gif** for serving plots to the WEB.

21. Smoothing and spline-fitting options for **plot**. See **plot smooth**.

22. **set margin** and **set origin** give much better control over where a graph appears on the page.

23. **set border** now controls each border individually.

24. The new commands **if** and **reread** allow command loops.

25. Point styles and sizes, line types and widths can be specified on the **plot** command. Line types and widths can also be specified for grids, borders, tics and arrows. See **plot with**. Furthermore these types may be combined and stored for further use. See **set linestyle**.

26. Text (labels, tic labels, and the time stamp) can be written vertically by those terminals capable of doing so.

5 Batch/Interactive Operation

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

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

Both the **exit** and **quit** commands terminate the current command file and **load** the next one, until all have been processed.

Examples:

To launch an interactive session:

```
gnuplot
```

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

```
gnuplot input1 input2
```

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

```
gnuplot header - trailer
```

6 Command-line-editing

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

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

The editing commands are as follows:

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, DEL`	delete the previous character.
`^D`	delete the current character.
`^K`	delete from current position to the end of line.
`^L, ^R`	redraw line in case it gets trashed.
`^U`	delete the entire line.
`^W`	delete from the current word to the end of line.
	History
`^P`	move back through history.
`^N`	move forward through history.

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

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

The Atari version of readline defines some additional key aliases:

Arrow key	Function
Undo	same as `^L`.
Home	same as `^A`.
Ctrl Home	same as `^E`.
Esc	same as `^U`.
Help	'**help**' plus return.
Ctrl Help	'**help** '.

7 Comments

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

8 Coordinates

The commands **set arrow**, **set key**, and **set label** allow you to draw something at an arbitrary position on the graph. This position is specified by the syntax:

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

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

first places the x, y, or z coordinate in the system defined by the left and bottom axes; **second** places it in the system defined by the second axes (top and right); **graph** specifies the area within the axes — 0,0 is bottom left and 1,1 is top right (for splot, 0,0,0 is bottom left of plotting area; use negative z to get to the base — see **set ticslevel**); and **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.

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.

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** and **set timefmt**. **gnuplot** will also accept an integer expression, which will be interpreted as seconds from 1 January 2000.

9 Environment

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

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

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

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

On Unix, HOME is used as the name of a directory to search for a .gnuplot file if none is found in the current directory. On AmigaOS, AtariTOS, MS-DOS and OS/2, gnuplot is used. On VMS, SYS$LOGIN: is used. See **help start-up**.

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

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

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

```
set PCTRM=S800
```

If PCTRM is not set, standard VGA is used.

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

10 Expressions

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

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

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

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

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

10.1 Functions

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

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

Math library functions		
Function	Arguments	Returns
abs(x)	any	absolute value of x, $\|x\|$; same type
abs(x)	complex	length of x, $\sqrt{\text{real}(x)^2 + \text{imag}(x)^2}$
acos(x)	any	$\cos^{-1} x$ (inverse cosine)
acosh(x)	any	$\cosh^{-1} x$ (inverse hyperbolic cosine) in radians
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
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
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
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
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
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)	any	rand(real(x)), pseudo random number generator
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 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

A few additional functions are also available.

other **gnuplot** functions		
Function	Arguments	Returns
column(x)	int	column x during datafile manipulation.
tm_hour(x)	int	the hour
tm_mday(x)	int	the day of the month
tm_min(x)	int	the minute
tm_mon(x)	int	the month
tm_sec(x)	int	the second
tm_wday(x)	int	the day of the week
tm_yday(x)	int	the day of the year
tm_year(x)	int	the year
valid(x)	int	test validity of column(x) during datafile manip.

10.2 Operators

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

Parentheses may be used to change order of evaluation.

10.2.1 Unary

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

Unary Operators		
Symbol	Example	Explanation
`-`	`-a`	unary minus
`+`	`+a`	unary plus (no-operation)
`~`	`~a`	* one's complement
`!`	`!a`	* logical negation
`!`	`a!`	* factorial
`$`	`$3`	* call arg/column during 'using' manipulation

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

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

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

10.2.2 Binary

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

Binary Operators		
Symbol	Example	Explanation
`**`	`a**b`	exponentiation
`*`	`a*b`	multiplication
`/`	`a/b`	division
`%`	`a%b`	* modulo
`+`	`a+b`	addition
`-`	`a-b`	subtraction
`==`	`a==b`	equality
`!=`	`a!=b`	inequality
`<`	`a<b`	less than
`<=`	`a<=b`	less than or equal to
`>`	`a>b`	greater than
`>=`	`a>=b`	greater than or equal to
`&`	`a&b`	* bitwise AND
`^`	`a^b`	* bitwise exclusive OR
`|`	`a|b`	* bitwise inclusive OR
`&&`	`a&&b`	* logical AND
`||`	`a||b`	* logical OR

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

10.2.3 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 )
```

Please see **plot data-file using** for an explanation of the **using** syntax.

10.3 User-defined

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

User-defined function syntax:

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

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

User-defined variable syntax:

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

Examples:

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

Note that the variable **pi** is already defined. But it is in no way magic; you may redefine it to be whatever you like.

Valid names are the same as in most programming languages: they must begin with a letter, but subsequent characters may be letters, digits, "$", or "_". Note, however, that the **fit** mechanism uses several variables with names that begin "FIT_". It is safest to avoid using such names. "FIT_LIMIT", however, is one that you may wish to redefine. See the documentation on **fit** for details.

See **show functions**, **show variables**, and **fit**.

11 Glossary

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

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

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

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

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

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

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

A graph may have up to four labelled axes. Various 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**. The names of the four axes for these usages are "x" for the axis along the bottom border of the plot, "y" for the left border, "x2" for the top border, and "y2" for the right border. "z" also occurs in commands used with 3-d plotting.

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

12 Plotting

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

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

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

splot can plot surfaces and contours in addition to points and/or lines. In addition to **splot**, see **set isosamples** for information about defining the grid for a 3-d function; **splot datafile** for information about the requisite file structure for 3-d data values; and **set contour** and **set cntrparam** for information about contours.

13 Start-up

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

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

14 Substitution

Command-line substitution is specified by a system command enclosed in backquotes. This command is spawned and the output it produces replaces the name of the command (and backquotes) on the command line. Some implementations also support pipes; see **plot data-file special-filenames**.

Newlines in the output produced by the spawned command are replaced with blanks.

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`"
```

15 Syntax

The general rules of syntax and punctuation in **gnuplot** are that keywords and options are order-dependent. Options and any accompanying parameters are separated by spaces whereas lists and coordinates are separated by commas. Ranges are separated by colons and enclosed in brackets [], text and file names are enclosed in quotes, and a few miscellaneous things are enclosed in parentheses. Braces {} are used for a few special purposes.

Commas are used to separate coordinates on the **set** commands **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.)

Brackets are used to delimit ranges, whether they are given on **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.

Braces are used in text to be specially processed by some terminals, like **postscript**. They are also used to denote complex numbers: $\{3,2\} = 3 + 2i$.

Text may be enclosed in single- or double-quotes. Backslash processing of sequences like \n (newline) and \345 (octal character code) is performed for double-quoted strings, but not for single-quoted strings.

The 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.
```

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.

At present you should not embed \n inside {} when using the enhanced option of the postscript terminal.

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

Back-quotes are used to enclose system commands for substitution.

16 Time/Date data

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

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

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

Commands like **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 give the deactivation command (like **set xdata**), the quantity will be shown in its numerical form.

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

If time/date information is to be plotted from a file, the **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 following example demonstrates time/date plotting.

Suppose the file "data" contains records like

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

This file can be plotted by

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

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

See the descriptions of each command for more details.

Part II
Commands

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

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

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

17 Cd

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

Syntax:

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

The directory name must be enclosed in quotes.

Examples:

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

DOS users *must* use single-quotes — backslash [\] has special significance inside double-quotes. For example,

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

fails, but

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

works as expected.

18 Call

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

The **call** command *must* be the last command on a multi-command line.

Syntax:

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

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

Example:

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

```
print "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:

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

NOTE: there is a clash in syntax with the datafile **using** callback operator. Use **$$n** or **column(n)** to access column n from a datafile inside a **call**ed datafile plot.

19 Clear

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

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)
set nomultiplot
```

Please see **set multiplot**, **set size**, and **set origin** for details of these commands.

20 Exit

The commands **exit** and **quit** and the END-OF-FILE character will exit the current **gnuplot** command file and **load** the next one. See "help batch/interactive" for more details.

Each of these commands will clear the output device (as does the **clear** command) before exiting.

21 Fit

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

Syntax:

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

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

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

analogous to **plot**; see **plot ranges**.

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

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

The default data formats for fitting functions with a single independent variable, y=f(x), are {x:}y or x:y:s; those formats can be changed with the datafile **using** qualifier. The third item, (a column number or an expression), if present, is interpreted as the standard deviation of the corresponding y value and is used to compute a weight for the datum, 1/s**2. Otherwise, all data points are weighted equally, with a weight of one.

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

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

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

Examples:

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

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

The fit may be interrupted by pressing Ctrl-C (any key but Ctrl-C under MSDOS and Atari Multitasking Systems). After the current iteration completes, you have the option to (1) stop the fit and accept the current parameter values, (2) continue the fit, (3) execute a **gnuplot** command as specified by the environment variable FIT_SCRIPT. The default for FIT_SCRIPT is **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 **update** command may be used to store final values in a file for subsequent use as a parameter file. See **update** for details.

21.1 Adjustable parameters

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

Adjustable parameters can be specified by a comma-separated list of variable names after the **via** keyword. Any variable that is not already defined is 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.

21.2 Beginner's guide

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 the sum of the squared differences or 'residuals' (SSR) between the input data points and the function values, evaluated at the same places. This quantity is often called 'chisquare' (i.e., the Greek letter chi, to the power of 2). The algorithm attempts to minimize SSR, or more precisely, WSSR, as the residuals are 'weighted' by the input data errors (or 1.0) before being squared; see **fit error_estimates** for details.

That's why it is called 'least-squares fitting'. Let's look at an example to see what is meant by 'non-linear', but first we had better go over some terms. Here it is convenient to use z as the dependent variable for user-defined functions of either one independent variable, z=f(x), or two independent variables, z=f(x,y). A parameter is a user-defined variable that **fit** will adjust, i.e., an unknown quantity in the function declaration. Linearity/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 criterium is met, either (1) the fit has "converged" (the relative change in WSSR is less than FIT_LIMIT), or (2) it reaches a preset iteration count limit, FIT_MAXITER (see **fit control variables**). The fit may also be interrupted and subsequently halted from the keyboard (see **fit**).

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

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

However, if all you really want is a smooth curve through your data points, the **smooth** option to **plot** may be what you've been looking for rather than **fit**.

21.3 Error estimates

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

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

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

21.3.1 Statistical overview

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

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

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

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

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

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

21.3.2 Practical guidelines

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

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

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

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

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

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

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

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

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

21.4 Fit controlling

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

21.4.1 Control variables

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

```
FIT_LIMIT
```

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

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

```
FIT_MAXITER
```

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

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

```
FIT_START_LAMBDA
```

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

```
FIT_LAMBDA_FACTOR
```

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

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

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

21.4.2 Environment variables

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

```
FIT_LOG
```

changes the name (and/or path) of the file to which the fit log will be written from the default of "fit.log" in the working directory.

```
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.

21.5 Multi-branch

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

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

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

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

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

21.6 Starting values

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

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

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

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

21.7 Tips

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

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

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

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

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

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

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

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

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

22 Help

The **help** command displays on-line help. To specify information on a particular topic use the syntax:

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

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

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

23 If

The **if** command allows commands to be executed conditionally.

Syntax:

```
if (<condition>) <command-line>
```

<condition> will be evaluated. If it is true (non-zero), then the command(s) of the <command-line> will be executed. If <condition> is false (zero), then the entire <command-line> is ignored. 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.

See **reread** for an example of how **if** and **reread** can be used together to perform a loop.

24 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 **comment** for information about comments in commands. To **load** with arguments, see **call**.

The **load** command *must* be the last command on a multi-command line.

Syntax:

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

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

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

Examples:

```
load 'work.gnu'
load "func.dat"
```

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.

25 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>"}
```

<time> may be any integer constant or expression. Choosing -1 will wait until a carriage return is hit, zero (0) won't pause at all, and a positive integer will wait the specified number of seconds. **pause 0** is synonymous with **print**.

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."
```

26 Plot

plot is the primary command for drawing plots with **gnuplot**. It creates plots of functions and data in many, many ways. **plot** is used to draw 2-d functions and data; **splot** draws 2-d projections of 3-d surfaces and data. **plot** and **splot** contain many common features; see **splot** for differences. Note specifically that **splot**'s **binary** and **matrix** options do not exist for **plot**.

Syntax:

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

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

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

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

Examples:

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

26.1 Data-file

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>' {index <index list>}
                      {every <every list>}
                      {thru <thru expression>}
                      {using <using list>}
                      {smooth <option>}
```

The modifiers **index**, **every**, **thru**, **using**, and **smooth** are discussed separately. In brief, **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** determines how the columns within a single record are to be interpreted (**thru** is a special case of **using**), and **smooth** allows for simple interpolation and approximation. ('splot' has a similar syntax, but does not support the **smooth** and **thru** options.)

Data files should contain at least one data point per record (**using** can select one data point from the record). Records beginning with # (and also with ! on VMS) will be treated as comments and ignored. Each data point represents an (x,y) pair. For **plot**s with error bars (see **set style errorbars**), each data point is (x,y,ydelta), (x,y,ylow,yhigh), (x,y,xdelta), (x,y,xlow,xhigh), or (x,y,xlow,xhigh,ylow,yhigh). In all cases, the numbers on each record of a data file must be separated by white space (one or more blanks or tabs), unless a format specifier is provided by the **using** option. This white space divides each record into columns.

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

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

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

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

If autoscaling has been enabled (**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 '-', '-'
1 1
```

```
19 19
e
1 1
19 19
e
```

26.1.1 Every

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

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

Syntax:

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

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

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

Note that records containing unplottable information are counted.

Any of the numbers can be omitted; the increments default to unity, the start values to the first point or block, and the end values to the last point or block. If **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
```

26.1.2 Example datafile

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

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

The file "population.dat" might contain:

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

26.1.3 Index

The **index** keyword allows only some of the data sets in a multi-data-set file to be plotted.

Syntax:

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

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

Example:

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

26.1.4 Smooth

gnuplot includes a few general-purpose routines for interpolation and approximation of data; 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 | csplines | acsplines | bezier | sbezier}
```

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

If **autoscale** is in effect, the ranges will be computed such that the plotted curve lies within the borders of the graph.

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

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

26.1.4.1 Acsplines

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

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

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

Example:

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

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

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

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

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

26.1.5 Special-filenames

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

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

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

10
12
14
e
2
4
6

10
12
14
e
```

does indeed work,

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

is a lot easier to type.

If you use **'-'** with **replot**, you may need to enter the data more than once (see **replot**).

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.)

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

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

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

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

While this approach is most flexible, it is possible to achieve simple filtering with the **using** or **thru** keywords.

26.1.6 Thru

The **thru** function is provided for backward compatibility.

Syntax:

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

It is equivalent to:

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

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

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

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

thru is parsed for **splot** and **fit** but has no effect.

26.1.7 Using

The most common datafile modifier is **using**.

Syntax:

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

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

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

N.B. — the **call** command also uses $'s as a special character. See **call** for details about how to include a column number in a **call** argument list.

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

'scanf' accepts several numerical specifications but **gnuplot** requires all inputs to be double-precision floating-point variables, so **lf** is the only permissible specifier. '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" or requires use of double-quotes rather than single-quotes.

Examples:

This creates a plot of the sum of the 2nd and 3rd data against the first: (The format string specifies comma- rather than space-separated columns.)

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

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

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

The meaning of this format is:

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

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

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

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

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

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

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

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

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

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

26.2 Errorbars

Error bars are supported for 2-d 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).

The error bars have crossbars at each end unless **set bar** is used (see **set bar** for details).

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

See **plot using**, **plot with**, and **set style** for more information.

26.3 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
```

26.4 Ranges

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

Syntax:

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

The first form applies to the independent variable (**xrange** or **trange**, if in parametric mode). The second form applies to the dependent variable **yrange** (and **xrange**, too, if in parametric mode). <dummy-var> is a new name for the independent variable. (The defaults may be changed with **set dummy**.) The optional <min> and <max> terms can be constant expressions or *.

In non-parametric mode, the order in which ranges must be given is **xrange** and **yrange**.

In parametric mode, the order for the **plot** command is **trange**, **xrange**, and **yrange**. The following **plot** command shows setting the **trange** to [-pi:pi], the **xrange** to [-1.3:1.3] and the **yrange** to [-1:1] for the duration of the graph:

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

Note that the x2range and y2range cannot be specified here — **set x2range** and **set y2range** must be used.

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

***** can be used to allow autoscaling of either of min and max. See also **set autoscale**.

Ranges specified on the **plot** or **splot** command line affect only that graph; use the **set xrange**, **set yrange**, etc., commands to change the default ranges for future graphs.

With time data, you must provide the range (in the same manner as the time appears in the datafile) within quotes. **gnuplot** uses the **timefmt** string to read the value — see **set timefmt**.

Examples:

This uses the current ranges:

```
plot cos(x)
```

This sets the x range only:

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

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

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

This sets both the x and y ranges:

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

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

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

This sets xmax and ymin only:

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

This sets the x range for a timeseries:

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

26.5 Title

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

Syntax:

```
title "<title>" | notitle
```

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

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

By default the line title is the function or file name as it appears on the **plot** command. If it is a file name, any datafile modifiers specified will be included in the default title.

The layout of the key itself (position, title justification, etc.) can be controlled by **set key**. Please see **set key** for details.

Examples:

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

```
plot x
```

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

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

This puts an untitled circular border around a polar graph:

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

26.6 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>}
                  {pointtype | pt <point_type>}
                  {pointsize | ps <point_size>}} }
```

where <style> is either **lines**, **points**, **linespoints**, **impulses**, **dots**, **steps**, **fsteps**, **histeps**, **errorbars**, **xerrorbars**, **yerrorbars**, **xyerrorbars**, **boxes**, **boxerrorbars**, **boxxyerrorbars**, **financebars**, **candlesticks** or **vector**. Some of these styles require additional information. See **set style <style>** for details of each style.

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

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

If you wish to choose the line or point type for a single plot, <line_type> and <point_type> may be specified. These are positive integer constants (or expressions) that specify the line type and point type to be used for the plot. Use **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** 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.

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

The keywords may be abbreviated as indicated.

Note that the **linewidth** and **pointsize** 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*y w points, x**2 + y**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
```

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 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
```

See **set style** to change the default styles.

27 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**.

28 Pwd

The **pwd** command prints the name of the working directory to the screen.

29 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.

30 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**, and similarly you can use **replot** to add a plot from a binary file only if the previous command was **splot**.

N.B. — use of

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

is not recommended. **gnuplot** does not store the inline data internally, so since **replot** appends new information to the previous **plot** and then executes the modified command, the **'-'** from the initial **plot** will expect to read inline data again.

Note that **replot** does not work in **multiplot** mode, since it reproduces only the last plot rather than the entire screen.

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

31 Reread

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

Examples:

Suppose the file "looper" contains the commands

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

and from within **gnuplot** you submit the commands

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

The result will be four plots (separated by the **pause** message).

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

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

and from within **gnuplot** you submit the commands

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

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

32 Reset

The **reset** command causes all options that can be set with the **set** command to take on their default values. The only exceptions are that the terminal set with **set term** and the output file set with **set output** are left unchanged. This command is useful, e.g., to restore the default settings at the end of a command file, or to return to a defined state after lots of settings have been changed within a command file. Please refer to the **set** command to see the default values that the various options take.

33 Save

The **save** command saves user-defined functions, variables, **set** options, or all three, plus the last **plot** (**splot**) command to the specified file.

Syntax:

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

where <option> is **functions**, **variables** or **set**. If no option is used, **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.

The filename must be enclosed in quotes.

Examples:

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

34 Set-show

The **set** command can be used to sets *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.

If a variable contains time/date data, **show** will display it according to the format currently defined by **set timefmt**, even if that was not in effect when the variable was initially defined.

34.1 Angles

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

34.2 Arrow

Arbitrary arrows can be placed on a plot using the **set arrow** command.

Syntax:

```
set arrow {<tag>} {from <position>} {to <position>} {{no}head}
          { {linestyle | ls <line_style>}
            | {linetype | lt <line_type>}
              {linewidth | lw <line_width} }
set noarrow {<tag>}
show arrow
```

<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>s are specified by either x,y or x,y,z, and may be preceded by **first**, **second**, **graph**, or **screen** to select the coordinate system. Unspecified coordinates default to 0. The endpoints can be specified in one of four coordinate systems — **first** or **second** axes, **graph** or **screen**. See **coordinates** for details. A coordinate system specifier does not carry over from the "from" position to the "to" position. Arrows outside the screen boundaries are permitted but may cause device errors.

Specifying **nohead** produces an arrow drawn without a head — a line segment. This gives you yet another way to draw a line segment on the plot. By default, arrows have heads.

The line style may be selected from a user-defined list of line styles (see **set linestyle**) 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 arrow** command with the appropriate index and **lt** or **lw**.

Examples:

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

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

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

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

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

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

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

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

To delete arrow number 2, use:

```
set noarrow 2
```

To delete all arrows, use:

```
set noarrow
```

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

```
show arrow
```

34.3 Autoscale

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

Syntax:

```
set autoscale {<axes>{min|max}}
set noautoscale {<axes>{min|max}}
show autoscale
```

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

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** for additional information about ranges.

The behavior of autoscaling remains consistent in parametric mode, (see **set parametric**). However, there are more dependent variables and hence more control over x, y, and z axis scales. In parametric mode, the independent or dummy variable is t for **plot**s and u,v for **splot**s. **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.

Autoscaling works the same way for polar mode as it does for parametric mode for **plot**, with the extension that in polar mode **set dummy** can be used to change the independent variable from t (see **set dummy**).

When tics are displayed on second axes but no plot has been specified for those axes, x2range and y2range are inherited from xrange and yrange. This is done *before* xrange and yrange are autoextended to a whole number of tics, which can cause unexpected results.

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 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:

```
set noautoscale
```

This disables autoscaling of the z axis only:

```
set noautoscale z
```

34.3.1 Parametric mode

When in parametric mode (**set parametric**), the xrange is as fully scalable as the y range. In other words, in parametric mode the x axis can be automatically scaled to fit the range of the parametric function that is being plotted. Of course, the y axis can also be automatically scaled just as in the non-parametric case. If autoscaling on the x axis is not set, the current x range is used.

Data files are plotted the same in parametric and non-parametric mode. However, there is a difference in mixed function and data plots: in non-parametric mode with autoscaled x, the x range of the datafile controls the x range of the functions; in parametric mode it has no influence.

For completeness a last command **set autoscale t** is accepted. However, the effect of this "scaling" is very minor. When **gnuplot** determines that the t range would be empty, it makes a small adjustment if autoscaling is true. Otherwise, **gnuplot** gives an error. Such behavior may, in fact, not be very useful and the command **set autoscale t** is certainly questionable.

splot extends the above ideas as you would expect. If autoscaling is set, then x, y, and z ranges are computed and each axis scaled to fit the resulting data.

34.3.2 Polar mode

When in polar mode (**set polar**), the xrange and the yrange are both found from the polar coordinates, and thus they can both be automatically scaled. In other words, in polar mode both the x and y axes can be automatically scaled to fit the ranges of the polar function that is being plotted.

When plotting functions in polar mode, the rrange may be autoscaled. When plotting data files in polar mode, the trange may also be autoscaled. Note that if the trange is contained within one quadrant, autoscaling will produce a polar plot of only that single quadrant.

Explicitly setting one or two ranges but not others may lead to unexpected results.

34.4 Bar

The **set bar** command controls the tics at the ends of errorbars.

Syntax:

```
set bar {small | large | <size>}
show bar
```

small is a synonym for 0.0, and **large** for 1.0. The default is 1.0 if no size is given.

34.5 Bmargin

The command **set bmargin** sets the size of the bottom margin. Please see **set margin** for details.

34.6 Border

The **set border** and **set noborder** commands control the display of the graph borders for the **plot** and **splot** commands.

Syntax:

```
set border {<integer> { {linestyle | ls <line_style>}
                        | {linetype | lt <line_type> }
                          {linewidth | lw <line_width>} } }
set noborder
show border
```

The borders are encoded in a 12-bit integer: the bottom four bits control the border for **plot** and the sides of the base for **splot**; The next four bits control the verticals in **splot**; the top four bits control the edges on top of the **splot**. In detail, the **<integer>** should be the sum of the appropriate entries from the following table:

Graph Border Encoding				
		Integer value of selection bit		
Side		plot border splot base	splot verticals	splot top
bottom	(south)	1	16	256
left	(west)	2	32	512
top	(north)	4	64	1024
right	(east)	8	128	2048

The default is 31, which is all four sides for **plot**, and base and z axis for **splot**.

Using the optional <line_style>, <line_type> and <line_width> specifiers, the way the border lines are drawn can be influenced (limited by what the current terminal driver supports). By default, the border is drawn with twice the usual linewidth. The <line_width> specifier scales this default value; for example, **set border 15 lw 2** will produce a border with four times the usual linewidth.

Various axes or combinations of axes may be added together in the command.

To have tics on edges other than bottom and left, disable the usual tics and enable the second axes.

Examples:

Draw all borders:
```
set border
```

Draw only the SOUTHWEST borders:
```
set border 3
```

Draw a complete box around a **splot**:
```
set border 4095
```

Draw a partial box, omitting the front vertical:
```
set border 127+256+512
```

Draw only the NORTHEAST borders:
```
set noxtics; set noytics; set x2tics; set y2tics; set border 12
```

34.7 Boxwidth

The **set boxwidth** command is used to set the default width of boxes in the **boxes** and **boxerrorbars** styles.

Syntax:
```
set boxwidth {<width>}
show boxwidth
```

If a data file is plotted without the width being specified in the third, fourth, or fifth column (or **using** entry), or if a function is plotted, the width of each box is set by the **set boxwidth** command. (If a width is given both in the file and by the **set boxwidth** command, the one in the file is used.) If the width is not specified in one of these ways, the width of each box will be calculated automatically so that it touches the adjacent boxes. 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 **set style boxerrorbars** for more details.

To set the box width to automatic use the command
```
set boxwidth
```

or, for four-column data,
```
set boxwidth -2
```

The same effect can be achieved with the **using** keyword in **plot**:
```
plot 'file' using 1:2:3:4:(-2)
```

34.8 Clabel

gnuplot will vary the linetype used for each contour level when clabel is set. When this option on (the default), a legend labels each linestyle with the z level it represents. It is not possible at present to separate the contour labels from the surface key.

Syntax:
```
set clabel {'<format>'}
set noclabel
show clabel
```

The default for the format string is %8.3g, which gives three decimal places. This may produce poor label alignment if the key is altered from its default configuration.

The first contour linetype, or only contour linetype when clabel is off, is the surface linetype +1; contour points are the same style as surface points.

See also **set contour**.

34.9 Clip

gnuplot can clip data points and lines that are near the boundaries of a graph.

Syntax:

```
set clip <clip-type>
set noclip <clip-type>
show clip
```

Three clip types are supported by **gnuplot**: **points**, **one**, and **two**. One, two, or all three clip types may be active for a single graph.

The **points** clip type forces **gnuplot** to clip (actually, not plot at all) data points that fall within but too close to the boundaries. This is done so that large symbols used for points will not extend outside the boundary lines. Without clipping points near the boundaries, the plot may look bad. Adjusting the x and y ranges may give similar results.

Setting the **one** clip type causes **gnuplot** to draw a line segment which has only one of its two endpoints within the graph. Only the in-range portion of the line is drawn. The alternative is to not draw any portion of the line segment.

Some lines may have both endpoints out of range, but pass through the graph. Setting the **two** clip-type allows the visible portion of these lines to be drawn.

In no case is a line drawn outside the graph.

The defaults are **noclip points**, **clip one**, and **noclip two**.

To check the state of all forms of clipping, use

```
show clip
```

For backward compatibility with older versions, the following forms are also permitted:

```
set clip
set noclip
```

set clip is synonymous with **set clip points**; **set noclip** turns off all three types of clipping.

34.10 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   auto {<n>} | <n>
                          | discrete <z1> {,<z2>{,<z3>...}}
                          | incremental <start>, <incr> {,<end>}
                  }
                }
show contour
```

This command has two functions. First, it sets the values of z for which contour points are to be determined (by linear interpolation between data points or function isosamples.) Second, it controls the way contours are drawn between the points determined to be of equal z. <n> should be an integral constant expression and <z1>, <z2> ... any constant expressions. The parameters are:

linear, **cubicspline**, **bspline** — Controls type of approximation or interpolation. If **linear**, then straight line segments connect points of equal z magnitude. If **cubicspline**, then piecewise-linear contours are interpolated between the same equal z points to form somewhat smoother contours, but which may undulate. If **bspline**, a guaranteed-smoother curve is drawn, which only approximates the position of the points of equal-z.

points — Eventually all drawings are done with piecewise-linear strokes. This number controls the number of line segments used to approximate the **bspline** or **cubicspline** curve. Number of cubicspline or bspline segments (strokes) = **points** * number of linear segments.

order — Order of the bspline approximation to be used. The bigger this order is, the smoother the resulting contour. (Of course, higher order bspline curves will move further away from the original piecewise linear data.) This option is relevant for **bspline** mode only. Allowed values are integers in the range from 2 (linear) to 10.

levels — Selection of contour levels, controlled by **auto** (default), **discrete**, **incremental**, and <n>, number of contour levels, limited to

```
MAX_DISCRETE_LEVELS as defined in plot.h (30 is standard.)
```

For **auto**, <n> specifies a nominal number of levels; the actual number will be adjusted to give simple labels. If the surface is bounded by zmin and zmax, contours will be generated at integer multiples of dz between zmin and zmax, where dz is 1, 2, or 5 times some power of ten (like the step between two tic marks).

For **levels discrete**, contours will be generated at z = <z1>, <z2> ... as specified; the number of discrete levels sets the number of contour levels. In **discrete** mode, any **set cntrparms levels <n>** are ignored.

For **incremental**, contours are generated at values of z beginning at <start> and increasing by <increment>, until the number of contours is reached. <end> is used to determine the number of contour levels, which will be changed by any subsequent **set cntrparam levels <n>**.

If the command **set cntrparam** is given without any arguments specified, the defaults are used: linear, 5 points, order 4, 5 auto levels.

Examples:

```
set cntrparam bspline
set cntrparam points 7
set cntrparam order 10
```

To select levels automatically, 5 if the level increment criteria are met:

```
set cntrparam levels auto 5
```

To specify discrete levels at .1, .37, and .9:

```
set cntrparam levels discrete .1,1/exp(1),.9
```

To specify levels from 0 to 4 with increment 1:

```
set cntrparam levels incremental  0,1,4
```

To set the number of levels to 10 (changing an incremental end or possibly the number of auto levels):

```
set cntrparam levels 10
```

To set the start and increment while retaining the number of levels:

```
set cntrparam levels incremental 100,50
```

See also **set contour** for control of where the contours are drawn, and **set clabel** for control of the format of the contour labels and linetypes.

34.11 Contour

set contour enables contour drawing for surfaces. This option is available for **splot** only.

Syntax:

```
set contour {base | surface | both}
set nocontour
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** for the parameters that affect the drawing of contours, and **set clabel** for control of labelling of the contours.

The surface can be switched off (see **set surface**), 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 file, and rereading it as a 2-d datafile plot:

```
set nosurface
set contour
set cntrparam ...
set term table
set out 'filename'
splot ...
set out
# contour info now in filename
set term <whatever>
plot 'filename'
```

In order to draw contours, the data should be organized as "grid data". In such a file all the points for a single y-isoline are listed, then all the points for the next y-isoline, and so on. A single blank line (a line containing no characters other than blank spaces and a carriage return and/or a line feed) separates one y-isoline from the next. See also **splot datafile**.

If contours are desired from non-grid data, **set dgrid3d** can be used to create an appropriate grid. See **set dgrid3d** for more information.

34.12 Data style

The **set data style** command changes the default plotting style for data plots.

Syntax:

```
set data style <style-choice>
show data style
```

See **set style** for the choices. If no choice is given, the choices are listed. **show data style** shows the current default data plotting style.

34.13 Dgrid3d

The **set dgrid3d** command enables, and can set parameters for, non-grid to grid data mapping.

Syntax:

```
set dgrid3d {<row_size>} {,{<col_size>} {,<norm>}}
set nodgrid3d
show dgrid3d
```

By default **dgrid3d** is disabled. When enabled, 3-d data read from a file are always treated as a scattered data set. A grid with dimensions derived from a bounding box of the scattered data and size as specified by the row/col_size parameters is created for plotting and contouring. The grid is equally spaced in x (rows) and in y (columns); the z values are computed as weighted averages of the scattered points' z values.

The third parameter, norm, controls the weighting: Each data point is weighted inversely by its distance from the grid point raised to the norm power. (Actually, the weights are given by the inverse of dx^norm + dy^norm, where dx and dy are the components of the separation of the grid point from each data point. For some norms that are powers of two, specifically 4, 8, and 16, the computation is optimized by using the Euclidean distance in the weight calculation, (dx^2+dx^2)^norm/2. However, any non-negative integer can be used.)

The closer the data point is to a grid point, the more effect it has on that grid point and the larger the value of norm the less effect more distant data points have on that grid point.

The **dgrid3d** option is a simple low pass filter that converts scattered data to a grid data set. More sophisticated approaches to this problem exist and should be used to preprocess the data outside **gnuplot** if this simple solution is found inadequate.

(The z values are found by weighting all data points, not by interpolating between nearby data points; also edge effects may produce unexpected and/or undesired results. In some cases, small norm values produce a grid point reflecting the average of distant data points rather than a local average, while large values of norm may produce "steps" with several grid points having the same value as the closest data point, rather than making a smooth transition between adjacent data points. Some areas of a grid may be filled by extrapolation, to an arbitrary boundary condition. The variables are not normalized; consequently the units used for x and y will affect the relative weights of points in the x and y directions.)

Examples:

```
set dgrid3d 10,10,1     # defaults
set dgrid3d ,,4
```

The first specifies that a grid of size 10 by 10 is to be constructed using a norm value of 1 in the weight computation. The second only modifies the norm, changing it to 4.

34.14 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)
```

At least one dummy variable must be set on the command; **set dummy** by itself will generate an error message.

Examples:

```
set dummy u,v
set dummy ,s
```

The second example sets the second variable to s.

34.15 Encoding

The **set encoding** command selects a character encoding. Valid values are **default**, which tells a terminal to use its default; **iso_8859_1** (known in the PostScript world as **ISO-Latin1**), which is used on many Unix workstations and with MS-Windows; **cp850**, for OS/2; and **cp437**, for MS-DOS.

Syntax:

```
set encoding {<value>}
show encoding
```

Note that encoding is not supported by all terminal drivers and that the device must be able to produce the desired non-standard characters.

34.16 Format

The format of the tic-mark labels can be set with the **set format** command.

Syntax:

```
set format {<axes>} {"<format-string>"}
set format {<axes>} {'<format-string>'}
show format
```

where <axes> is either **x**, **y**, **z**, **xy**, **x2**, **y2** or nothing (which is the same as **xy**). The length of the string representing a tic mark (after formatting with 'printf') is restricted to 100 characters. If the format string is omitted, the format will be returned to the default "%g". For LaTeX users, the format "$%g$" is often desirable. If the empty string "" is used, no label will be plotted with each tic, though the tic mark will still be plotted. To eliminate all tic marks, use **set noxtics** or **set noytics**.

Newline (\n) is accepted in the format string. Use double-quotes rather than single-quotes to enable such interpretation. See also **syntax**.

The default format for both axes is "%g", but other formats such as "%.2f" or "%3.0em" are often desirable. Anything accepted by 'printf' when given a double precision number, and accepted by the terminal, will work. Some other options have been added. If the format string looks like a floating point format, then **gnuplot** tries to construct a reasonable format.

Characters not preceded by "%" are printed verbatim. Thus you can include spaces and labels in your format string, such as "%g m", which will put " m" after each number. If you want "%" itself, double it: "%g %%".

See also **set xtics** for more information about tic labels.

34.16.1 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
%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
%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; "#", 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 releases of 'printf' 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 "%.0P%%"; 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** for a list of the allowed input format codes.

34.16.2 Time/date specifiers

In time/date mode, the acceptable formats are:

Tic-mark label Date/Time Format 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, 1–31
%D	shorthand for "%m/%d/%y"
%H or %k	hour, 0–24
%I or %l	hour, 0–12
%j	day of the year, 1–366
%m	month, 1–12
%M	minute, 0–60
%p	"am" or "pm"
%r	shorthand for "%I:%M:%S %p"
%R	shorthand for %H:%M"
%S	second, 0–60
%T	shorthand for "%H:%M:%S"
%U	week of the year (week starts on Sunday)
%w	day of the week, 0–6 (Sunday = 0)
%W	week of the year (week starts on Monday)
%y	year, 0-99
%Y	year, 4-digit

Except for the non-numerical formats, these may be preceded by a "0" ("zero", not "oh") to pad the field length with leading zeroes, and a positive digit, to define the minimum field width (which will be overridden if the specified width is not large enough to contain the number). There is a 24-character limit to the length of the printed text; longer strings will be truncated.

Examples:

Suppose the text is "76/12/25 23:11:11". Then

```
set format x                 # defaults to "12/25/76" \n "23:11"
set format x "%A, %d %b %Y"  # "Saturday, 25 Dec 1976"
set format x "%r %d"         # "11:11:11 pm 12/25/76"
```

Suppose the text is "98/07/06 05:04:03". Then

```
set format x "%1y/%2m/%3d %01H:%02M:%03S"  # "98/ 7/  6 5:04:003"
```

34.17 Function style

The **set function style** command changes the default plotting style for function plots.

Syntax:

```
set function style <style-choice>
show function style
```

See **set style** for the choices. If no choice is given, the choices are listed. **show function style** shows the current default function plotting style.

34.18 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**.

34.19 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}
         {polar {<angle>}}
         { {linestyle <major_linestyle>}
           | {linetype | lt <major_linetype>}
             {linewidth | lw <major_linewidth>}
           { , {linestyle | ls <minor_linestyle>}
               | {linetype | lt <minor_linetype>}
                 {linewidth | lw <minor_linewidth>} } }
set nogrid
show grid
```

The grid can be enabled and disabled for the major and/or minor tic marks on any axis, and the linetype and linewidth can be specified for major and minor grid lines, also via a predefined linestyle, as far as the active terminal driver supports this.

Additionally, a polar grid can be selected for 2-d plots — circles are drawn to intersect the selected tics, and radial lines are drawn at definable intervals. (The interval is given in degrees or radians ,depending on the **set angles** setting.) Note that a polar grid is no longer automatically generated in polar mode.

The pertinent tics must be enabled before **set grid** can draw them; **gnuplot** will quietly ignore instructions to draw grid lines at non-existent tics, but they will appear if the tics are subsequently enabled.

If no linetype is specified for the minor gridlines, the same linetype as the major gridlines is used. The default polar angle is 30 degrees.

By default, grid lines are drawn with half the usual linewidth. The major and minor linewidth specifiers scale this default value; for example, **set grid lw .5** will draw grid lines with one quarter the usual linewidth.

Z grid lines are drawn on the back of the plot. This looks better if a partial box is drawn around the plot — see **set border**.

34.20 Hidden3d

The **set hidden3d** command enables hidden line removal for surface plotting (see **splot**). Some optional features of the underlying algorithm can also be controlled using this command.

Syntax:
```
set hidden3d {defaults} |
             { {{offset <offset>} | {nooffset}}
               {trianglepattern <bitpattern>}
               {{undefined <level>} | {noundefined}}
               {{no}altdiagonal}
               {{no}bentover} }
set nohidden3d
show hidden3d
```

In contrast to the usual display in gnuplot, hidden line removal actually treats the given function or data grids as real surfaces that can't be seen through, so parts behind the surface will be hidden by it.

For this to be possible, the surface needs to have 'grid structure' (see **splot datafile** 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**) 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. Labels and arrows are always visible and are unaffected. The key is also never hidden by the 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** and **set isosamples** for discussion of the difference.

The algorithm used to remove the hidden parts of the surfaces has some additional features controllable by this command. Specifying **defaults** will set them all to their default settings, as detailed below. If **defaults** is not given, only explicitly specified options will be influenced: all others will keep their previous values, so you can turn on/off hidden line removal via **set {no}hidden3d**, without modifying the set of options you chose.

The first option, **offset**, influences the linestyle used for lines on the 'back' side. Normally, they are drawn in a linestyle one index number higher than the one used for the front, to make the two sides of the surface distinguishable. You can specify a different line style offset to add instead of the default 1, by **offset <offset>**. Option **nooffset** stands for **offset 0**, making the two sides of the surface use the same linestyle.

Next comes the option **trianglepattern <bitpattern>**. <bitpattern> must be a number between 0 and 7, interpreted as a bit pattern. Each bit determines the visibility of one edge of the triangles each surface is split up into. Bit 0 is for the 'horizontal' edges of the grid, Bit 1 for the 'vertical' ones, and Bit 2 for the diagonals that split each cell of the original grid into two triangles. The default pattern is 3, making all horizontal and vertical lines visible, but not the diagonals. You may want to choose 7 to see those diagonals as well.

The **undefined <level>** option lets you decide what the algorithm is to do with data points that are undefined (missing data, or undefined function values), or exceed the given x-, y- or z-ranges. Such points can either be plotted nevertheless, or taken out of the input data set. All surface elements touching a point that is taken out will be taken out as well, thus creating a hole in the surface. If <level> = 3, equivalent to option **noundefined**, no points will be thrown away at all. This may produce all kinds of problems elsewhere, so you should avoid this. <level> = 2 will throw away undefined points, but keep the out-of-range ones. <level> = 1, the default, will get rid of out-of-range points as well.

By specifying **noaltdiagonal**, you can override the default handling of a special case can occur if **undefined** is active (i.e. <level> is not 3). Each cell of the grid-structured input surface will be divided in two triangles along one of its diagonals. Normally, all these diagonals have the same orientation relative to the grid. If exactly one of the four cell corners is excluded by the **undefined** handler, and this is on the usual diagonal, both triangles will be excluded. However if the default setting of **altdiagonal** is active, the other diagonal will be chosen for this cell instead, minimizing the size of the hole in the surface.

The **bentover** option controls what happens to another special case, this time in conjunction with the **trianglepattern**. For rather crumply surfaces, it can happen that the two triangles a surface cell is divided into are seen from opposite sides (i.e. the original quadrangle is 'bent over'), as illustrated in the following ASCII art:

```
                                                            C----B
  original quadrangle:  A--B      displayed quadrangle:     |\   |
    ("set view 0,0")    | /|    ("set view 75,75" perhaps)  | \  |
                        |/ |                                |  \ |
                        C--D                                |   \|
```

```
A     D
```

If the diagonal edges of the surface cells aren't generally made visible by bit 2 of the <bitpattern> there, the edge CB above wouldn't be drawn at all, normally, making the resulting display hard to understand. Therefore, the default option of **bentover** will turn it visible in this case. If you don't want that, you may choose **nobentover** instead.

34.21 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** and **set hidden3d**. 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.

34.22 Key

The **set key** enables a key (or legend) describing plots on a plot.

The contents of the key, i.e., the names given to each plotted data set and function and samples of the lines and/or symbols used to represent them, are determined by the **title** and **with** options of the {s}**plot** command. Please see **plot title** and **plot with** for more information.

Syntax:

```
set key {  left | right | top | bottom | outside | below
         | <position>}
        {Left | Right} {{no}reverse}
        {samplen <sample_length>} {spacing <vertical_spacing>}
        {width <width_increment>}
        {title "<text>"}
        {{no}box { {linestyle | ls <line_style>}
                 | {linetype | lt <line_type>}
                   {linewidth | lw <line_width>}}}
set nokey
show key
```

By default the key is placed in the upper right corner of the graph. The keywords **left**, **right**, **top**, **bottom**, **outside** and **below** may be used to place the key in the other corners inside the graph or to the right (outside) or below the graph. They may be given alone or combined.

Justification of the labels within the key is controlled by **Left** or **Right** (default is **Right**). The text and sample can be reversed (**reverse**) and a box can be drawn around the key (**box {...}**) in a specified **linetype** and **linewidth**, or a user-defined **linestyle**. Note that not all terminal drivers support linewidth selection, though.

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 it is not drawn. <sample_length> must be an integer.

The vertical spacing between lines is controlled by **spacing**. The spacing is set equal to the product of the pointsize, the vertical tic size, and <vertical_spacing>. The program will guarantee that the vertical spacing is no smaller than the character height.

The <width_increment> is a number of character widths to be added to or subtracted from the length of the string. This is useful only when you are putting a box around the key and you are using control characters in the text. **gnuplot** simply counts the number of characters in the string when computing the box width; this allows you to correct it.

A title can be put on the key (**title "<text>"**) — see also **syntax** for the distinction between text in single- or double-quotes. The key title uses the same justification as do the plot titles.

The defaults for **set key** are **right**, **top**, **Right**, **noreverse**, **samplen 4**, **spacing 1.25**, **title ""**, and **nobox**. The default <linetype> is the same as that used for the plot borders. Entering **set key** with no options returns the key to its default configuration.

The <position> can be a simple x,y,z as in previous versions, but these can be preceded by one of four keywords (**first**, **second**, **graph**, **screen**) which selects the coordinate system in which the position is specified. See **coordinates** for more details.

The key is drawn as a sequence of lines, with one plot described on each line. On the right-hand side (or the left-hand side, if **reverse** is selected) of each line is a representation that attempts to mimic the way the curve is plotted. On the other side of each line is the text description (the line title), obtained from the **plot** command. The lines are vertically arranged so that an imaginary straight line divides the left- and right-hand sides of the key. It is the coordinates of the top of this line that are specified with the **set key** command. In a **plot**, only the x and y coordinates are used to specify the line position. For a **splot**, x, y and z are all used as a 3-d location mapped using the same mapping as the graph itself to form the required 2-d screen position of the imaginary line.

Some or all of the key may be outside of the graph boundary, although this may interfere with other labels and may cause an error on some devices. If you use the keywords **outside** or **below**, **gnuplot** makes space for the keys and the graph becomes smaller. Putting keys outside to the right, they occupy as few columns as possible, and putting them below, as many columns as possible (depending of the length of the labels), thus stealing as little space from the graph as possible.

When using the TeX or PostScript drivers, or similar drivers where formatting information is embedded in the string, **gnuplot** is unable to calculate correctly the width of the string for key positioning. If the key is to be positioned at the left, it may be convenient to use the combination **set key left Left reverse**. The box and gap in the grid will be the width of the literal string.

If **splot** is being used to draw contours, the contour labels will be listed in the key. If the alignment of these labels is poor or a different number of decimal places is desired, the label format can be specified. See **set clabel** for details.

Examples:

This places the key at the default location:

```
set key
```

This disables the key:

```
set nokey
```

This places a key at coordinates 2,3.5,2 in the default (first) coordinate system:

```
set key 2,3.5,2
```

This places the key below the graph:

```
set key below
```

This places the key in the bottom left corner, left-justifies the text, gives it a title, and draws a box around it in linetype 3:

```
set key left bottom Left title 'Legend' box 3
```

34.23 Label

Arbitrary labels can be placed on the plot using the **set label** command.

Syntax:

```
set label {<tag>} {"<label_text>"} {at <position>}
          {<justification>} {{no}rotate} {font "<name><,size>"}
set nolabel {<tag>}
show label
```

The <position> is specified by either x,y or x,y,z, and may be preceded by **first**, **second**, **graph**, or **screen** to select the coordinate system. See **coordinates** for details.

The tag is an integer that is used to identify the label. If no <tag> is given, the lowest unused tag value is assigned automatically. The tag can be used to delete or modify a specific label. To change any attribute of an existing label, use the **set label** command with the appropriate tag, and specify the parts of the label to be changed.

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 parameter <justification>, which may be **left**, **right** or **center**, indicating that the point is to be at the left, right or center of the text. Labels outside the plotted boundaries are permitted but may interfere with axis labels or other text.

If **rotate** is given, the label is written vertically (if the terminal can do so, of course).

If 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** and **set timefmt**.

The EEPIC, Imagen, LaTeX, and TPIC drivers allow \\ in a string to specify a newline.

Examples:

To set a label at (1,2) to "y=x", use:

```
set label "y=x" at 1,2
```

To set a Sigma of size 24, from the Symbol font set, at the center of the graph, use:

```
set label "S" at graph 0.5,0.5 center font "Symbol,24"
```

To set a label "y=x^2" with the right of the text at (2,3,4), and tag the label as number 3, use:

```
set label 3 "y=x^2" at 2,3,4 right
```

To change the preceding label to center justification, use:

```
set label 3 center
```

To delete label number 2, use:

```
set nolabel 2
```

To delete all labels, use:

```
set nolabel
```

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
```

34.24 Linestyle

Each terminal has a default set of line and point types, which can be seen by using the command **test**. **set linestyle** 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 linestyle <index> {linetype | lt <line_type>}
                      {linewidth | lw <line_width>}
                      {pointtype | pt <point_type>}
                      {pointsize | ps <point_size>}
set nolinestyle
show linestyle
```

The line and point types are taken from the default types for the terminal currently in use. The line width and point size are multipliers for the default width and size (but note that <point_size> here is unaffected by the multiplier given on 'set pointsize').

The defaults for the line and point types is the index. The defaults for the width and size are both unity.

Linestyles created by this mechanism do not replace the default styles; both may be used.

Not all terminals support the **linewidth** and **pointsize** features; if not supported, the option will be ignored.

Note that this feature is not completely implemented; linestyles defined by this mechanism may be used with 'plot', 'splot', 'replot', and 'set arrow', but not by other commands that allow the default index to be used, such as 'set grid'.

Example: 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 linestyle 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 function style 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 function style linespoints
plot p(x) lt 1 pt 3, q(x) ls 1
```

will create a plot of f(x) using the default triangles connected by a red line and q(x) using small triangles connected by a green line.

34.25 Lmargin

The command **set lmargin** sets the size of the left margin. Please see **set margin** for details.

34.26 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 default value is determined from the LANG environment variable.

34.27 Logscale

Log scaling may be set on the x, y, z, x2 and/or y2 axes.

Syntax:

```
set logscale <axes> <base>
set nologscale <axes>
show logscale
```

where <axes> may be any combinations of **x**, **y**, and **z**, in any order, or **x2** or **y2** and where <base> is the base of the log scaling. If <base> is not given, then 10 is assumed. If <axes> is not given, then all axes are assumed. **set nologscale** turns off log scaling for the specified axes.

Examples:

To enable log scaling in both x and z axes:

```
set logscale xz
```

To enable scaling log base 2 of the y axis:

```
set logscale y 2
```

To disable z axis log scaling:

```
set nologscale z
```

34.28 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 polar and azimuthal angles theta and phi (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.

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**.

34.29 Margin

The computed margins can be overridden by the **set margin** commands. **show margin** shows the current settings.

Syntax:

```
set bmargin {<margin>}
set lmargin {<margin>}
set rmargin {<margin>}
set tmargin {<margin>}
show margin
```

The units of <margin> are character heights or widths, as appropriate. A positive value defines the absolute size of the margin. A negative value (or none) causes **gnuplot** to revert to the computed value.

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.

34.30 Missing

The **set missing** command allows you to tell **gnuplot** what character is used in a data file to denote missing data.

Syntax:

```
set missing {"<character>"}
show missing
```

Example:

```
set missing "?"
```

would mean that, when plotting a file containing

```
1 1
2 ?
3 2
```

the middle line would be ignored.

There is no default character for **missing**.

34.31 Multiplot

The command **set multiplot** places **gnuplot** in the multiplot mode, in which several plots are placed on the same page, window, or screen.

Syntax:

```
set multiplot
set nomultiplot
```

For some terminals, no plot is displayed until the command **set nomultiplot** 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 a plot, but the screen may not be cleared between plots.

Any labels or arrows that have been defined will be drawn for each plot according to the current size and origin (unless their coordinates are defined in the **screen** system). Just about everything else that can be **set** is applied to each plot, too. If you want something to appear only once on the page, for instance a single time stamp, you'll need to put a **set time**/**set notime** pair around one of the **plot**, **splot** or **replot** commands within the **set multiplot**/**set nomultiplot** block.

The commands **set origin** and **set size** must be used to correctly position each plot; see **set origin** and **set size** for details of their usage.

Example:

```
set size 0.7,0.7
set origin 0.1,0.1
set multiplot
set size 0.4,0.4
set origin 0.1,0.1
plot sin(x)
set size 0.2,0.2
set origin 0.5,0.5
plot cos(x)
set nomultiplot
```

displays a plot of cos(x) stacked above a plot of sin(x). Note the initial **set size** and **set origin**. While these are not always required, their inclusion is recommended. Some terminal drivers require that bounding box information be available before any plots can be made, and the form given above guarantees that the bounding box will include the entire plot array rather than just the bounding box of the first plot.

set size and **set origin** refer to the entire plotting area used for each plot. If you want to have the axes themselves line up, you can guarantee that the margins are the same size with the **set margin** commands. See **set margin** 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.

34.32 Mx2tics

Minor tic marks along the x2 (top) axis are controlled by **set mx2tics**. Please see **set mxtics**.

34.33 Mxtics

Minor tic marks along the x axis are controlled by **set mxtics**. They can be turned off with **set nomxtics**. Similar commands control minor tics along the other axes.

Syntax:

```
set mxtics {<freq> | default}
set nomxtics
show mxtics
```

The same syntax applies to **mytics**, **mztics**, **mx2tics** and **my2tics**.

<freq> is the number of sub-intervals (NOT the number of minor tics) between major tics (ten is the default for a linear axis, so there are nine 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.

Minor tics can be used only with uniformly spaced major tics. Since major tics can be placed arbitrarily by **set {x|x2|y|y2|z}tics**, minor tics cannot be used if major tics are explicitly **set**.

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** for information about these.

34.34 My2tics

Minor tic marks along the y2 (right-hand) axis are controlled by **set my2tics**. Please see **set mxtics**.

34.35 Mytics

Minor tic marks along the y axis are controlled by **set mytics**. Please see **set mxtics**.

34.36 Mztics

Minor tic marks along the z axis are controlled by **set mztics**. Please see **set mxtics**.

34.37 Offsets

Offsets provide a mechanism to put a boundary around the data inside of an autoscaled graph.

Syntax:

```
set offsets <left>, <right>, <top>, <bottom>
set nooffsets
show offsets
```

Each offset may be a constant or an expression. Each defaults to 0. Left and right offsets are given in units of the x axis, top and bottom offsets in units of the y axis. A positive offset expands the graph in the specified direction, e.g., a positive bottom offset makes ymin more negative. Negative offsets, while permitted, can have unexpected interactions with autoscaling and clipping.

Offsets are ignored in **splot**s.

Example:

```
set offsets 0, 0, 2, 2
plot sin(x)
```

This graph of sin(x) will have a y range [-3:3] because the function will be autoscaled to [-1:1] and the vertical offsets are each two.

34.38 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** for information about this system).

Syntax:

```
set origin <x-origin>,<y-origin>
```

34.39 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**, ignore **set output**.])

MSDOS users should note that the \ character has special significance in double-quoted strings, so single-quotes should be used for filenames in different directories.

When both **set terminal** and **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 to know whether or not a file is to be formatted in order to open it properly.

On machines with popen functions (Unix), output can be piped through a shell command if the first non-whitespace character of the filename is '|'. For instance,

```
set output "|lpr -Plaser filename"
set output "|lp -dlaser filename"
```

On MSDOS machines, **set output "PRN"** will direct the output to the default printer. On VMS, output can be sent directly to any spooled device. It is also possible to send the output to DECnet transparent tasks, which allows some flexibility.

34.40 Parametric

The **set parametric** command changes the meaning of **plot** (**splot**) from normal functions to parametric functions. The command **set noparametric** restores the plotting style to normal, single-valued expression plotting.

Syntax:

```
set parametric
set noparametric
show parametric
```

For 2-d plotting, a parametric function is determined by a pair of parametric functions operating on a parameter. An example of a 2-d parametric function would be **plot sin(t),cos(t)**, which draws a circle (if the aspect ratio is set correctly — see **set size**). **gnuplot** will display an error message if both functions are not provided for a parametric **plot**.

For 3-d plotting, the surface is described as x=f(u,v), y=g(u,v), z=h(u,v). Therefore a triplet of functions is required. An example of a 3-d parametric function would be **cos(u)*cos(v),cos(u)*sin(v),sin(u)**, which draws a sphere. **gnuplot** will display an error message if all three functions are not provided for a parametric **splot**.

The total set of possible plots is a superset of the simple f(x) style plots, since the two functions can describe the x and y values to be computed separately. In fact, plots of the type t,f(t) are equivalent to those produced with f(x) because the x values are computed using the identity function. Similarly, 3-d plots of the type u,v,f(u,v) are equivalent to f(x,y).

Note that the order the parametric functions are specified is xfunction, yfunction (and zfunction) and that each operates over the common parametric domain.

Also, the **set parametric** function implies a new range of values. Whereas the normal f(x) and f(x,y) style plotting assume an xrange and yrange (and zrange), the parametric mode additionally specifies a trange, urange, and vrange. These ranges may be set directly with **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.

34.41 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** for details.

Please note that the pointsize setting is not supported by all terminal types.

34.42 Polar

The **set polar** command changes the meaning of the plot from rectangular coordinates to polar coordinates.

Syntax:

```
set polar
set nopolar
show polar
```

There have been changes made to polar mode in version 3.7, so that scripts for **gnuplot** versions 3.5 and earlier will require modification. The main change is that the dummy variable t is used for the angle so that the x and y ranges can be controlled independently. Other changes are: 1) tics are no longer put along the zero axes automatically — use **set xtics axis nomirror**; **set ytics axis nomirror**; 2) the grid, if selected, is not automatically polar — use **set grid polar**; 3) the grid is not labelled with angles — use **set label** as necessary.

In polar coordinates, the dummy variable (t) is an angle. The default range of t is [0:2*pi], or, if degree units have been selected, to [0:360] (see **set angles**).

The command **set nopolar** 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** command for similar functionality for **splot**s.

While in polar coordinates the meaning of an expression in t is really r = f(t), where t is an angle of rotation. The trange controls the domain (the angle) of the function, and the x and y ranges control the range of the graph in the x and y directions. Each of these ranges, as well as the rrange, may be autoscaled or set explicitly. See **set xrange** for details of all the **set range** commands.

Example:

```
set polar
```

```
plot t*sin(t)
plot [-2*pi:2*pi] [-3:3] [-3:3] t*sin(t)
```

The first **plot** uses the default polar angular domain of 0 to 2*pi. The radius and the size of the graph are scaled automatically. The second **plot** expands the domain, and restricts the size of the graph to [-3:3] in both directions.

You may want to **set size square** to have **gnuplot** try to make the aspect ratio equal to unity, so that circles look circular.

34.43 Rmargin

The command **set rmargin** sets the size of the right margin. Please see **set margin** for details.

34.44 Rrange

The **set rrange** command sets the range of the radial coordinate for a graph in polar mode. Please see **set xrange** for details.

34.45 Samples

The sampling rate of functions, or for interpolating data, may be changed by the **set samples** command.

Syntax:

```
set samples <samples_1> {,<samples_2>}
show samples
```

By default, sampling is set to 100 points. A higher sampling rate will produce more accurate plots, but will take longer. This parameter has no effect on data file plotting unless one of the interpolation/approximation options is used. See **plot smooth** re 2-d data and **set cntrparam** and **set dgrid3d** re 3-d data.

When a 2-d graph is being done, only the value of <samples_1> is relevant.

When a surface plot is being done without the removal of hidden lines, the value of samples specifies the number of samples that are to be evaluated for the isolines. Each iso-v line will have <sample_1> samples and each iso-u line will have <sample_2> samples. If you only specify <samples_1>, <samples_2> will be set to the same value as <samples_1>. See also **set isosamples**.

34.46 Size

The **set size** command scales the displayed size of the plot.

Syntax:

```
set size {{no}square | ratio <r> | noratio} {<xscale>,<yscale>}
show size
```

The <xscale> and <yscale> values are the scaling factors for the size of the plot, which includes the graph and the margins.

ratio causes **gnuplot** to try to create a graph with an aspect ratio of <r> (the ratio of the y-axis length to the x-axis length) within the portion of the plot specified by <xscale> and <yscale>.

The meaning of a negative value for <r> is different. If <r>=-1, gnuplot tries to set the scales so that the unit has the same length on both the x and y axes (suitable for geographical data, for instance). If <r>=-2, the unit on y has twice the length of the unit on x, and so on.

The success of **gnuplot** in producing the requested aspect ratio depends on the terminal selected. The graph area will be the largest rectangle of aspect ratio <r> that will fit into the specified portion of the output (leaving adequate margins, of course).

square is a synonym for **ratio 1**.

Both **noratio** and **nosquare** return the graph to the default aspect ratio of the terminal, but do not return <xscale> or <yscale> to their default values (1.0).

ratio and **square** have no effect on 3-d plots.

set size is relative to the default size, which differs from terminal to terminal. Since **gnuplot** fills as much of the available plotting area as possible by default, it is safer to use **set size** to decrease the size of a plot than to increase it. See **set terminal** for the default sizes.

On some terminals, changing the size of the plot will result in text being misplaced.

Examples:

To set the size to normal size use:
```
set size 1,1
```

To make the graph half size and square use:
```
set size square 0.5,0.5
```

To make the graph twice as high as wide use:
```
set size ratio 2
```

34.47 Style

Default styles are chosen with the **set function style** and **set data style** commands. See **plot with** for information about how to override the default plotting style for individual functions and data sets.

Syntax:
```
set function style <style>
set data style <style>
show function style
show data style
```

The types used for all line and point styles (i.e., solid, dash-dot, color, etc. for lines; circles, squares, crosses, etc. for points) will be either those specified on the **plot** or **splot** command or will be chosen sequentially from the types available to the terminal in use. Use the command **test** to see what is available.

None of the styles requiring more than two columns of information (e.g., **errorbars**) can be used with **splot**s or function **plot**s. Neither **boxes** nor any of the **steps** styles can be used with **splot**s. If an inappropriate style is specified, it will be changed to **points**.

For 2-d data with more than two columns, **gnuplot** is picky about the allowed **errorbar** styles. The **using** option on the **plot** command can be used to set up the correct columns for the style you want. (In this discussion, "column" will be used to refer both to a column in the data file and an entry in the **using** list.)

For three columns, only **xerrorbars**, **yerrorbars** (or **errorbars**), **boxes**, and **boxerrorbars** are allowed. If another plot style is used, the style will be changed to **yerrorbars**. The **boxerrorbars** style will calculate the boxwidth automatically.

For four columns, only **xerrorbars**, **yerrorbars** (or **errorbars**), **xyerrorbars**, **boxxyerrorbars**, and **boxerrorbars** are allowed. An illegal style will be changed to **yerrorbars**.

Five-column data allow only the **boxerrorbars**, **financebars**, and **candlesticks** styles. (The last two of these are primarily used for plots of financial prices.) An illegal style will be changed to **boxerrorbars** before plotting.

Six- and seven-column data only allow the **xyerrorbars** and **boxxyerrorbars** styles. Illegal styles will be changed to **xyerrorbars** before plotting.

For more information about error bars, please see **plot errorbars**.

34.47.1 Boxerrorbars

The **boxerrorbars** style is only relevant to 2-d data plotting. It is a combination of the **boxes** and **yerrorbars** styles. The boxwidth will come from the fourth column if the y errors are in the form of "ydelta" and the boxwidth was not previously set equal to -2.0 (**set boxwidth -2.0**) or from the fifth column if the y errors are in the form of "ylow yhigh". The special case **boxwidth = -2.0** is for four-column data with y errors in the form "ylow yhigh". In this case the boxwidth will be calculated so that each box touches the adjacent boxes. The width will also be calculated in cases where three-column data are used.

The box height is determined from the y error in the same way as it is for the **yerrorbars** style — either from y-ydelta to y+ydelta or from ylow to yhigh, depending on how many data columns are provided.

34.47.2 Boxes

The **boxes** style is only relevant to 2-d plotting. It draws a box centered about the given x coordinate from the x axis (not the graph border) to the given y coordinate. The width of the box is obtained in one of three ways. If it is a data plot and the data file has a third column, this will be used to set the width of the box. If not, if a width has been set using the **set boxwidth** command, this will be used. If neither of these is available, the width of each box will be calculated automatically so that it touches the adjacent boxes.

34.47.3 Boxxyerrorbars

The **boxxyerrorbars** style is only relevant to 2-d data plotting. It is a combination of the **boxes** and **xyerrorbars** styles.

The box width and height are determined from the x and y errors in the same way as they are for the **xyerrorbars** style — either from xlow to xhigh and from ylow to yhigh, or from x-xdelta to x+xdelta and from y-ydelta to y+ydelta , depending on how many data columns are provided.

34.47.4 Candlesticks

The **candlesticks** style is only relevant for 2-d data plotting of financial data. Five columns of data are required; in order, these should be the x coordinate (most likely a date) and the opening, low, high, and closing prices. The symbol is an open rectangle, 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 width of the rectangle may be changed by **set bar**. The symbol will be unchanged if the low and high prices are interchanged or if the opening and closing prices are interchanged. See **set bar** and **financebars**.

34.47.5 Dots

The **dots** style plots a tiny dot at each point; this is useful for scatter plots with many points.

34.47.6 Financebars

The **financebars** style is only relevant for 2-d data plotting of financial data. Five columns of data are required; in order, these should be the x coordinate (most likely a date) and the opening, low, high, and closing prices. The symbol is a vertical line segment, located horizontally at the x coordinate and limited vertically by the high and low prices. A horizontal tic on the left marks the opening price and one on the right marks the closing price. The length of these tics may be changed by **set bar**. The symbol will be unchanged if the high and low prices are interchanged. See **set bar** and **candlesticks**.

34.47.7 Fsteps

The **fsteps** style is only relevant to 2-d plotting. It connects consecutive points with two line segments: the first from (x1,y1) to (x1,y2) and the second from (x1,y2) to (x2,y2).

34.47.8 Histeps

The **histeps** style is only relevant to 2-d plotting. It is intended for plotting histograms. Y-values are assumed to be centered at the x-values; the point at x1 is represented as a horizontal line from ((x0+x1)/2,y1) to ((x1+x2)/2,y1). The lines representing the end points are extended so that the step is centered on at x. Adjacent points are connected by a vertical line at their average x, that is, from ((x1+x2)/2,y1) to ((x1+x2)/2,y2).

If **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.

histeps is only a plotting style; **gnuplot** does not have the ability to create bins and determine their population from some data set.

34.47.9 Impulses

The **impulses** style displays a vertical line from the x axis (not the graph border), or from the grid base for **splot**, to each point.

34.47.10 Lines

The **lines** style connects adjacent points with straight line segments.

34.47.11 Linespoints

The **linespoints** style does both **lines** and **points**, that is, it draws a small symbol at each point and then connects adjacent points with straight line segments. The command **set pointsize** may be used to change the size of the points. See **set pointsize** for its usage.

linespoints may be abbreviated **lp**.

34.47.12 Points

The **points** style displays a small symbol at each point. The command **set pointsize** may be used to change the size of the points. See **set pointsize** for its usage.

34.47.13 Steps

The **steps** style is only relevant to 2-d plotting. It connects consecutive points with two line segments: the first from (x1,y1) to (x2,y1) and the second from (x2,y1) to (x2,y2).

34.47.14 Vector

The **vector** style draws a vector from (x,y) to (x+xdelta,y+ydelta). Thus it requires four columns of data. It also draws a small arrowhead at the end of the vector.

set clip one and **set clip two** affect drawing vectors. Please see **set clip**.

34.47.15 Xerrorbars

The **xerrorbars** style is only relevant to 2-d data plots. **xerrorbars** is like **dots**, except that a horizontal error bar is also drawn. At each point (x,y), a line is drawn from (xlow,y) to (xhigh,y) or from (x-xdelta,y) to (x+xdelta,y), depending on how many data columns are provided. A tic mark is placed at the ends of the error bar (unless **set bar** is used — see **set bar** for details).

34.47.16 Xyerrorbars

The **xyerrorbars** style is only relevant to 2-d data plots. **xyerrorbars** is like **dots**, except that horizontal and vertical error bars are also drawn. At each point (x,y), lines are drawn from (x,y-ydelta) to (x,y+ydelta) and from (x-xdelta,y) to (x+xdelta,y) or from (x,ylow) to (x,yhigh) and from (xlow,y) to (xhigh,y), depending upon the number of data columns provided. A tic mark is placed at the ends of the error bar (unless **set bar** is used — see **set bar** for details).

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
```

34.47.17 Yerrorbars

The **yerrorbars** (or **errorbars**) style is only relevant to 2-d data plots. **yerrorbars** is like **dots**, except that a vertical error bar is also drawn. At each point (x,y), a line is drawn from (x,y-ydelta) to (x,y+ydelta) or from (x,ylow) to (x,yhigh), depending on how many data columns are provided. A tic mark is placed at the ends of the error bar (unless **set bar** is used — see **set bar** for details).

34.48 Surface

The command **set surface** controls the display of surfaces by **splot**.

Syntax:

```
set surface
set nosurface
show surface
```

The surface is drawn with the style specifed by **with**, or else the appropriate style, data or function.

Whenever **set nosurface** is issued, **splot** will not draw points or lines corresponding to the function or data file points. Contours may be still be drawn on the surface, depending on the **set contour** option. **set nosurface; set contour base** is useful for displaying contours on the grid base. See also **set contour**.

34.49 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>}
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.

Several terminals have additional options. For example, see **dumb**, **iris4d**, **hpljii** or **postscript**.

This document may describe drivers that are not available to you because they were not installed, or it may not describe all the drivers that are available to you, depending on its output format.

34.49.1 Aed767

The **aed512** and **aed767** terminal drivers support AED graphics terminals. The two drivers differ only in their horizontal ranges, which are 512 and 768 pixels, respectively. Their vertical range is 575 pixels. There are no options for these drivers.

34.49.2 Aifm

Several options may be set in **aifm** — the Adobe Illustrator 3.0+ driver.

Syntax:
```
set terminal aifm {<color>} {"<fontname>"} {<fontsize>}
```

<color> is either **color** or **monochrome**; "<fontname>" is the name of a valid PostScript font; <fontsize> is the size of the font in PostScript points, before scaling by the **set size** command. Selecting **default** sets all options to their default values: **monochrome**, "Helvetica", and 14pt.

Since AI does not really support multiple pages, multiple graphs will be drawn directly on top of one another. However, each graph will be grouped individually, making it easy to separate them inside AI (just pick them up and move them).

Examples:
```
set term aifm
set term aifm 22
set size 0.7,1.4; set term aifm color "Times-Roman" 14
```

34.49.3 Amiga

The **amiga** terminal, for Commodore Amiga computers, allows the user to plot either to a screen (default), or, if Kickstart 3.0 or higher is installed, to a window on the current public screen. The font and its size can also be selected.

Syntax:
```
set terminal amiga {screen | window} {"<fontname>"} {<fontsize>}
```

The default font is 8-point "topaz".

The screen option uses a virtual screen, so it is possible that the graph will be larger than the screen.

34.49.4 Apollo

The **apollo** terminal driver supports the Apollo Graphics Primitive Resource with rescaling after window resizing. It has no options.

If a fixed-size window is desired, the **gpr** terminal may be used instead.

34.49.5 Atari ST (via AES)

The **atari** terminal has options to set the character size and the screen colors.

Syntax:

```
set terminal atari {<fontsize>} {<col0> <col1> ... <col15.}
```

The character size must appear if any colors are to be specified. Each of the (up to 16) colors is given as a three-digit hex number, where the digits represent RED, GREEN and BLUE (in that order). The range of 0–15 is scaled to whatever color range the screen actually has. On a normal ST screen, odd and even intensities are the same.

Examples:

```
set terminal atari 4    # use small (6x6) font
set terminal atari 6 0  # set monochrome screen to white on black
set terminal atari 13 0 fff f00 f0 f ff f0f
           # set first seven colors to black, white, green, blue,
           # cyan, purple, and yellow and use large font (8x16).
```

Additionally, if an environment variable GNUCOLORS exists, its contents are interpreted as an options string, but an explicit terminal option takes precedence.

34.49.6 Atari ST (via VDI)

The **vdi** terminal is the same as the **atari** terminal, except that it sends output to the screen via the VDI and not into AES-Windows.

The **vdi** terminal has options to set the character size and the screen colors.

Syntax:

```
set terminal vdi {<fontsize>} {<col0> <col1> ... <col15.}
```

The character size must appear if any colors are to be specified. Each of the (up to 16) colors is given as a three-digit hex number, where the digits represent RED, GREEN and BLUE (in that order). The range of 0–15 is scaled to whatever color range the screen actually has. On a normal ST screen, odd and even intensities are the same.

Examples:

```
set terminal vdi 4    # use small (6x6) font
set terminal vdi 6 0  # set monochrome screen to white on black
set terminal vdi 13 0 fff f00 f0 f ff f0f
           # set first seven colors to black, white, green, blue,
           # cyan, purple, and yellow and use large font (8x16).
```

Additionally, if an environment variable GNUCOLORS exists, its contents are interpreted as an options string, but an explicit terminal option takes precedence.

34.49.7 Be

gnuplot provides the **be** terminal type for use with X servers. This terminal type is set automatically at startup if the **DISPLAY** environment variable is set, if the **TERM** environment variable is set to **xterm**, or if the **-display** command line option is used.

Syntax:

```
    set terminal be {reset} {<n>}
```

Multiple plot windows are supported: **set terminal be <n>** directs the output to plot window number n. If n>0, the terminal number will be appended to the window title and the icon will be labeled **gplt <n>**. The active window may distinguished by a change in cursor (from default to crosshair.)

Plot windows remain open even when the **gnuplot** driver is changed to a different device. A plot window can be closed by pressing the letter q while that window has input focus, or by choosing **close** from a window manager menu. All plot windows can be closed by specifying **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.

34.49.7.1 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.
'-tvtwm'	requests that geometry specifications for position of the window be made relative to the currently displayed portion of the virtual root.
'-raise'	raise plot window after each plot.
'-noraise'	do 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 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.

34.49.7.2 Monochome_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.

34.49.7.3 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
```

34.49.7.4 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

34.49.7.5 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
```

34.49.8 Cgi

The **cgi** and **hcgi** terminal drivers support SCO CGI drivers. **hcgi** is for printers; the environment variable CGIPRNT must be set. **cgi** may be used for either a display or hardcopy; if the environment variable CGIDISP is set, then that display is used. Otherwise CGIPRNT is used.

These terminals have no options.

34.49.9 Cgm

The **cgm** terminal generates a Computer Graphics Metafile, Version 1. This file format is a subset of the ANSI X3.122-1986 standard entitled "Computer Graphics - Metafile for the Storage and Transfer of Picture Description Information". Several options may be set in **cgm**.

Syntax:

```
set terminal cgm {<mode>} {<color>} {<rotation>} {solid | dashed}
                 {width <plot_width>} {linewidth <line_width>}
                 {"<font>"} {<fontsize>}
                 {<color0> <color1> <color2> ...}
```

where <mode> is **landscape**, **portrait**, or **default**; <color> is either **color** or **monochrome**; <rotation> is either **rotate** or **norotate**; **solid** draws all curves with solid lines, overriding any dashed patterns; <plot_width> is the assumed width of the plot in points; <line_width> is the line width in points (default 1); <font> is the name of a font; and <fontsize> is the size of the font in points (default 12).

By default, **cgm** uses rotated text for the Y axis label.

The first six options can be in any order. Selecting **default** sets all options to their default values.

Each color must be of the form 'xrrggbb', where x is the literal character 'x' and 'rrggbb' are the red, green and blue components in hex. For example, 'x00ff00' is green. The background color is set first, then the plotting colors. Examples:

```
set terminal cgm landscape color rotate dashed width 432 \
               linewidth 1  'Helvetica Bold' 12       # defaults
set terminal cgm 14 linewidth 2  14  # wider lines & larger font
set terminal cgm portrait "Times Italic" 12
set terminal cgm color solid      # no pesky dashes!
```

34.49.9.1 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 16 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
Helvetica Bold
Helvetica Oblique
Helvetica Bold Oblique
Times Roman
Times Bold
Times Italic
Times Bold Italic
Courier
Courier Bold
Courier Oblique
Courier Bold Oblique
Symbol
Hershey/Cartographic_Roman
Hershey/Cartographic_Greek
Hershey/Simplex_Roman
Hershey/Simplex_Greek
Hershey/Simplex_Script
Hershey/Complex_Roman
Hershey/Complex_Greek
Hershey/Complex_Script
Hershey/Complex_Italic
Hershey/Complex_Cyrillic
Hershey/Duplex_Roman
Hershey/Triplex_Roman
Hershey/Triplex_Italic
Hershey/Gothic_German
Hershey/Gothic_English
Hershey/Gothic_Italian
Hershey/Symbol_Set_1
Hershey/Symbol_Set_2
Hershey/Symbol_Math
ZapfDingbats
Script
15

The first thirteen of these fonts are required for WebCGM. The Microsoft Office CGM import filter implements the 13 standard fonts listed above, and also 'ZapfDingbats' and 'Script'. However, the script font may only be accessed under the name '15'. For more on Microsoft import filter font substitutions, check its help file which you may find here:

```
C:\Program Files\Microsoft Office\Office\Cgmimp32.hlp
```

and/or its configuration file, which you may find here:

```
C:\Program Files\Common Files\Microsoft Shared\Grphflt\Cgmimp32.cfg
```

In the **set term** command, you may specify a font name which does not appear in the default font table. In that case, a new font table is constructed with the specified font as its first entry. You must ensure that the spelling, capitalization, and spacing of the name are appropriate for the application that will read the CGM file. (Gnuplot and any MIL-D-28003A compliant application ignore case in font names.) If you need to add several new fonts, use several **set term** commands.

Example:

```
set terminal cgm 'Old English'
set terminal cgm 'Tengwar'
set terminal cgm 'Arabic'
set output 'myfile.cgm'
plot ...
set output
```

You cannot introduce a new font in a **set label** command.2 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.

34.49.9.2 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

34.49.9.3 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.

34.49.9.4 Solid The **solid** option may be used to disable dashed line styles in the plots. This is useful when color is enabled and the dashing of the lines detracts from the appearance of the plot. The **dashed** option confirms the default behavior, which gives a different dash pattern to each curve.

34.49.9.5 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.

34.49.9.6 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
```

34.49.9.7 Nofontlist The default font table includes the fonts recommended for WebCGM, which are compatible with the Computer Graphics Metafile input filter for Microsoft Office and Corel Draw. Another application might use different fonts and/or different font names, which may not be documented. As a workaround, the **nofontlist** option deletes the font table from the CGM file. In this case, the reading application should use a default table. Gnuplot will still use its own default font table to select font indices. Thus, 'Helvetica' will give you an index of 1, which should get you the first entry in your application's default font table. 'Helvetica Bold' will give you its second entry, etc.

The former **winword6** option is now a deprecated synonym for **nofontlist**. The problems involving the color and font tables that the **winword6** option was intended to work around turned out to be gnuplot bugs which have now been fixed.

34.49.10 Corel

The **corel** terminal driver supports CorelDraw.

Syntax:

```
set terminal corel {  default
                    | {monochrome | color
                         {<fontname> {"<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.

34.49.11 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.

34.49.12 Svga

The **svga** terminal driver supports PCs with SVGA graphics. It can only be be used if it is compiled with DJGPP. Its only option is the font.

Syntax:

```
set terminal svga {"<fontname>"}
```

34.49.13 Dumb

The **dumb** terminal driver has an optional size specification and trailing linefeed control.

Syntax:

```
set terminal dumb {[no]feed} {<xsize> <ysize>}
```

where <xsize> and <ysize> set the size of the dumb terminals. Default is 79 by 24. The last newline is printed only if **feed** is enabled.

Examples:

```
set term dumb nofeed
set term dumb 79 49 # VGA screen---why would anyone do that?
```

34.49.14 Dxf

The **dxf** terminal driver creates pictures that can be imported into AutoCad (Release 10.x). It has no options of its own, but some features of its plots may be modified by other means. The default size is 120x80 AutoCad units, which can be changed by **set size**. **dxf** uses seven colors (white, red, yellow, green, cyan, blue and magenta), which can be changed only by modifying the source file. If a black-and-white plotting device is used, the colors are mapped to differing line thicknesses. See the description of the AutoCad print/plot command.

34.49.15 Dxy800a

This terminal driver supports the Roland DXY800A plotter. It has no options.

34.49.16 Eepic

The **eepic** terminal driver supports the extended LaTeX picture environment. It is an alternative to the **latex** driver.

The output of this terminal is intended for use with the "eepic.sty" macro package for LaTeX. To use it, you need "eepic.sty", "epic.sty" and a printer driver that supports the "tpic" \specials. If your printer driver doesn't support those \specials, "eepicemu.sty" will enable you to use some of them. dvips and dvipdfm do support the "tpic" \specials.

Syntax:

```
set terminal eepic {color, dashed, rotate, small, tiny, default, <fontsize>}
```

Options: You can give options in any order you wish. 'color' causes gnuplot to produce \color{...} commands so that the graphs are colored. Using this option, you must include \usepackage{color} in the preambel of your latex document. 'dashed' will allow dashed line types; without this option, only solid lines with varying thickness will be used. 'dashed' and 'color' are mutually exclusive; if 'color' is specified, then 'dashed' will be ignored 'rotate' will enable true rotated text (by 90 degrees). Otherwise, rotated text will be typeset with letters stacked above each other. If you use this option you must include \usepackage{graphicx} in the preamble. 'small' will use \scriptsize symbols as point markers (Probably does not work with TeX, only LaTeX2e). Default is to use the default math size. 'tiny' uses \scriptscriptstyle symbols. 'default' resets all options to their defaults = no color, no dashed lines, pseudo-rotated (stacked) text, large point symbols. <fontsize> is a number which specifies the font size inside the picture environment; the unit is pt (points), i.e., 10 pt equals approx. 3.5 mm. If fontsize is not specified, then all text inside the picture will be set in \footnotesize.

Notes: Remember to escape the # character (or other chars meaningful to (La-)TeX) by \\ (2 backslashes). It seems that dashed lines become solid lines when the vertices of a plot are too close. (I do not know if that is a general problem with the tpic specials, or if it is caused by a bug in eepic.sty or dvips/dvipdfm.) The default size of an eepic plot is 5x3 inches, which can be scaled by 'set size a,b' Points, among other things, are drawn using the LaTeX commands "\Diamond", "\Box", etc. These commands no longer belong to the LaTeX2e core; they are included in the latexsym package, which is part of the base distribution and thus part of any LaTeX implementation. Please do not forget to use this package. Instead of latexsym, you can also include the amssymb package. All drivers for LaTeX offer a special way of controlling text positioning: If any text string begins with '{', you also need to include a '}' at the end of the text, and the whole text will be centered both horizontally and vertically. If the text string begins with '[', you need to follow this with a position specification (up to two out of t,b,l,r), ']{', the text itself, and finally '}'. The text itself may be anything LaTeX can typeset as an LR-box. '\rule{}{}'s may help for best positioning.

Examples: set term eepic

```
output graphs as eepic macros inside a picture environment;
\input the resulting file in your LaTeX document.
```

set term eepic color tiny rotate 8

```
eepic macros with \color macros, \scripscriptsize point markers,
true rotated text, and all text set with 8pt.
```

About label positioning: Use gnuplot defaults (mostly sensible, but sometimes not really best):

```
set title '\LaTeX\ -- $ \gamma $'
```

Force centering both horizontally and vertically:

```
set label '{\LaTeX\ -- $ \gamma $}' at 0,0
```

Specify own positioning (top here):

```
set xlabel '[t]{\LaTeX\ -- $ \gamma $}'
```

The other label – account for long ticlabels:

```
set ylabel '[r]{\LaTeX\ -- $ \gamma $\rule{7mm}{0pt}'
```

34.49.17 Emf

The **emf** terminal generates an Enhanced Metafile Format file. This file format is the metafile standard on MS Win32 Systems Syntax:

```
set terminal emf {<color>} {solid | dashed}
                 {"<font>"} {<fontsize>}
```

<color> is either **color** or **monochrome**; **solid** draws all curves with solid lines, overriding any dashed patterns; <font> is the name of a font; and <**fontsize**> is the size of the font in points.

The first two options can be in any order. Selecting **default** sets all options to their default values.

Examples:

```
set terminal emf 'Times Roman Italic' 12
set terminal emf color solid    # no pesky dashes!
```

34.49.18 Emxvga

The **emxvga**, **emxvesa** and **vgal** terminal drivers support PCs with SVGA, vesa SVGA and VGA graphics boards, respectively. They are intended to be compiled with "emx-gcc" under either DOS or OS/2. They also need VESA and SVGAKIT maintained by Johannes Martin (JMARTIN@GOOFY.ZDV.UNI-MAINZ.DE) with additions by David J. Liu (liu@phri.nyu.edu).

Syntax:

```
set terminal emxvga
set terminal emxvesa {vesa-mode}
set terminal vgal
```

The only option is the vesa mode for **emxvesa**, which defaults to G640x480x256.

34.49.19 Epslatex

Two options may be set in the **epslatex** driver.

Syntax:

```
set terminal epslatex {default}
                      {color | monochrome} {solid | dashed}
                      {"<fontname>"} {<fontsize>}
```

default mode sets all options to their defaults: **monochrome**, **dashed**, "default" and 11pt.

`Default size of a plot is 5 inches wide and 3 inches high.`

solid draws all plots with solid lines, overriding any dashed patterns; "<**fontname**>" is the name of font; and <**fontsize**> is the size of the font in PostScript points. Font selection isn't supported yet. Font size selection is supported only for the calculation of proper spacing. The actual LaTeX font 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 documents, use '"default" 12' as options.

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. See also the documenation of the pslatex terminal driver. To create multiline labels, use \shortstack, example

```
set ylabel '[r]{\shortstack{first line \\ second line}}'
```

The driver produces two different files, one for the LaTeX part and one for the eps part of the figure. The name of the LaTeX file is derived from the name of the eps file given on the **set output** command; it is determined by replacing the trailing **.eps** (actually just the final extent in the file name — and the option will be turned off if there is no extent) with **.tex** in the output file name. Remember to close the file before leaving **gnuplot**. There is no LaTeX output if no output file is given! In your LaTeX documents use '\input{filename}' for inclusion of the figure. Include \usepackage{graphics} in the preambel! Via 'epstopdf' (contained e.g. in the teTeX package, requires ghostscript) pdf files can made out of the eps files. If the graphics package is properly configured, the LaTeX files can also be processed by pdflatex without changes, and the pdf files are included instead of the eps files

34.49.20 Epson-180dpi

This driver supports a family of Epson printers and derivatives.

epson-180dpi and **epson-60dpi** are drivers for Epson LQ-style 24-pin printers with resolutions of 180 and 60 dots per inch, respectively.

epson-lx800 is a generic 9-pin driver appropriate for printers like the Epson LX-800, the Star NL-10 and NX-1000, the PROPRINTER, and so forth.

nec-cp6 is generix 24-pin driver that can be used for printers like the NEC CP6 and the Epson LQ-800.

The **okidata** driver supports the 9-pin OKIDATA 320/321 Standard printers.

The **starc** driver is for the Star Color Printer.

The **tandy-60dpi** driver is for the Tandy DMP-130 series of 9-pin, 60-dpi printers.

Only **nec-cp6** has any options.

Syntax:

```
set terminal nec-cp6 {monochrome | colour | draft}
```

which defaults to monochrome.

With each of these drivers, a binary copy is required on a PC to print. Do not use **print** — use instead **copy file /b lpt1:**.

34.49.21 Excl

The **excl** terminal driver supports Talaris printers such as the EXCL Laser printer and the 1590. It has no options.

34.49.22 Hercules

These drivers supports PC monitors with autodetected graphics boards. They can be used only when compiled with Zortech C/C++. None have options.

34.49.23 Fig

The **fig** terminal device generates output in the Fig graphics language.

Syntax:

```
set terminal fig {monochrome | color} {small | big}
                 {pointsmax <max_points>}
                 {landscape | portrait}
                 {metric | inches}
                 {fontsize <fsize>}
                 {size <xsize> <ysize>}
                 {thickness <units>}
                 {depth <layer>}
```

monochrome and **color** determine whether the picture is black-and-white or **color**. **small** and **big** produce a 5x3 or 8x5 inch graph in the default **landscape** mode and 3x5 or 5x8 inches in **portrait** mode. <max_points> sets the maximum number of points per polyline. Default units for editing with "xfig" may be **metric** or **inches**. **fontsize** sets the size of the text font to <fsize> points. **size** sets (overrides) the size of the drawing area to <xsize>*<ysize> in units of inches or centimeters depending on the **inches** or **metric** setting in effect. **depth** sets the default depth layer for all lines and text. The default depth is 10 to leave room for adding material with "xfig" on top of the plot.

thickness sets the default line thickness, which is 1 if not specified. Overriding the thickness can be achieved by adding a multiple of 100 to the to the **linetype** value for a **plot** command. In a similar way the **depth** of plot elements (with respect to the default depth) can be controlled by adding a multiple of 1000 to <linetype>. The depth is then <layer> + <linetype>/1000 and the thickness is (<linetype>%1000)/100 or, if that is zero, the default line thickness.

Additional point-plot symbols are also available with the **fig** driver. The symbols can be used through **pointtype** values % 100 above 50, with different fill intensities controlled by <pointtype> % 5 and outlines in black (for <pointtype> % 10 < 5) or in the current color. Available symbols are

```
50 - 59:  circles
60 - 69:  squares
70 - 79:  diamonds
80 - 89:  upwards triangles
90 - 99:  downwards triangles
```

The size of these symbols is linked to the font size. The depth of symbols is by default one less than the depth for lines to achieve nice error bars. If <pointtype> is above 1000, the depth is <layer> + <pointtype>/1000-1. If <pointtype>%1000 is above 100, the fill color is (<pointtype>%1000)/100-1.

Available fill colors are (from 1 to 9): black, blue, green, cyan, red, magenta, yellow, white and dark blue (in monochrome mode: black for 1 to 6 and white for 7 to 9).

See **plot with** for details of <linetype> and <pointtype>.

The **big** option is a substitute for the **bfig** terminal in earlier versions, which is no longer supported.

Examples:

```
set terminal fig monochrome small pointsmax 1000  # defaults

plot 'file.dat' with points linetype 102 pointtype 759
```

would produce circles with a blue outline of width 1 and yellow fill color.

```
plot 'file.dat' using 1:2:3 with err linetype 1 pointtype 554
```

would produce errorbars with black lines and circles filled red. These circles are one layer above the lines (at depth 9 by default).

To plot the error bars on top of the circles use

```
plot 'file.dat' using 1:2:3 with err linetype 1 pointtype 2554
```

34.49.24 Ggi

The GGI terminal generates output to a GGI target.

34.49.25 Gif

The **gif** terminal driver generates output in GIF format. It uses Thomas Boutell's gd library, which is available from http://www.boutell.com/gd/

By default, the **gif** terminal driver uses a shared Web-friendy palette. Syntax:

```
set terminal gif {transparent} {interlace}
                 {tiny | small | medium | large | giant}
                 {size <x>,<y>}
                 {<color0> <color1> <color2> ...}
```

transparent instructs the driver to generate transparent GIFs. The first color will be the transparent one.

interlace instructs the driver to generate interlaced GIFs.

The choice of fonts is **tiny** (5x8 pixels), **small** (6x12 pixels), **medium** (7x13 Bold), **large** (8x16) or **giant** (9x15 pixels)

The size <x,y> is given in pixels — it defaults to 640x480. The number of pixels can be also modified by scaling with the **set size** command.

Each color must be of the form 'xrrggbb', where x is the literal character 'x' and 'rrggbb' are the red, green and blue components in hex. For example, 'x00ff00' is green. The background color is set first, then the border colors, then the X & Y axis colors, then the plotting colors. The maximum number of colors that can be set is 256.

Examples:

```
set terminal gif small size 640,480 \
                 xffffff x000000 x404040 \
                 xff0000 xffa500 x66cdaa xcdb5cd \
                 xadd8e6 x0000ff xdda0dd x9500d3    # defaults
```

which uses white for the non-transparent background, black for borders, gray for the axes, and red, orange, medium aquamarine, thistle 3, light blue, blue, plum and dark violet for eight plotting colors.

```
set terminal gif transparent xffffff \
                 x000000 x202020 x404040 x606060 \
                 x808080 xA0A0A0 xC0C0C0 xE0E0E0 \
```

which uses white for the transparent background, black for borders, dark gray for axes, and a gray-scale for the six plotting colors.

The page size is 640x480 pixels. The **gif** driver can create either color or monochromatic output, but you have no control over which is produced.

The current version of the **gif** driver does not support animated GIFs.

34.49.26 Unixplot

The **unixplot** driver produces device-independent output in the GNU plot graphics language. The default size of the PostScript results generated by "plot2ps" is 5 x 3 inches; this can be increased up to about 8.25 x 8.25 by **set size**.

Syntax:

```
set terminal unixplot {"<fontname>"} {<fontsize>}
```

which defaults to 10-point "Courier".

There is a non-GNU version of the **unixplot** driver which cannot be compiled unless this version is left out.

34.49.27 Gpic

The **gpic** terminal driver generates GPIC graphs in the Free Software Foundations's "groff" package. The default size is 5 x 3 inches. The only option is the origin, which defaults to (0,0).

Syntax:

```
set terminal gpic {<x> <y>}
```

where **x** and **y** are in inches.

A simple graph can be formatted using

```
groff -p -mpic -Tps file.pic > file.ps.
```

The output from pic can be pipe-lined into eqn, so it is possible to put complex functions in a graph with the **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 the command

```
set terminal gpic x y
```

for example, using

```
.PS 6.0
copy "trig.pic"
.PE
```

34.49.28 Gpr

The **gpr** terminal driver supports the Apollo Graphics Primitive Resource for a fixed-size window. It has no options.

If a variable window size is desired, use the **apollo** terminal instead.

34.49.29 Grass

The **grass** terminal driver gives **gnuplot** capabilities to users of the GRASS geographic information system. Contact grassp-list@moon.cecer.army.mil for more information. Pages are written to the current frame of the GRASS Graphics Window. There are no options.

34.49.30 Hp2648

The **hp2648** terminal driver supports the Hewlett Packard HP2647 and HP2648. It has no options.

34.49.31 Hp2623a

The **hp2623a** terminal driver supports the Hewlett Packard HP2623A. It has no options.

34.49.32 Hp500c

The **hp500c** terminal driver supports the Hewlett Packard HP DeskJet 500c. It has options for resolution and compression.

Syntax:

```
set terminal hp500c {<res>} {<comp>}
```

where **res** can be 75, 100, 150 or 300 dots per inch and **comp** can be "rle", or "tiff". Any other inputs are replaced by the defaults, which are 75 dpi and no compression. Rasterization at the higher resolutions may require a large amount of memory.

34.49.33 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** for details).

Syntax:

```
set terminal hpgl {<number_of_pens>} {eject}
```

The selection

```
set terminal hpgl 8 eject
```

is equivalent to the previous **hp7550** terminal, and the selection

```
set terminal hpgl 4
```

is equivalent to the previous **hp7580b** terminal.

The **pcl5** driver supports plotters such as the Hewlett-Packard Designjet 750C, the Hewlett-Packard Laserjet III, and the Hewlett-Packard Laserjet IV. It actually uses HPGL-2, but there is a name conflict among the terminal devices. It has several options which must be specified in the order indicated below:

Syntax:

```
set terminal pcl5 {mode <mode>} {<plotsize>}
    {{color {<number_of_pens>}} | monochrome} {solid | dashed}
    {font <font>} {size <fontsize>} {pspoints | nopspoints}
```

<mode> is **landscape** or **portrait**. <plotsize> is the physical plotting size of the plot, which is one of the following: **letter** for standard (8 1/2" X 11") displays, **legal** for (8 1/2" X 14") displays, **noextended** for (36" X 48") displays (a letter size ratio) or, **extended** for (36" X 55") displays (almost a legal size ratio). **color** is for multi-pen (i.e. color) plots, and <number_of_pens> is the number of pens (i.e. colors) used in color plots. **monochrome** is for one (e.g. black) pen plots. **solid** draws all lines as solid lines, or 'dashed' will draw lines with different dashed and dotted line patterns. <font> is **stick**, **univers**, **cg_times**, **zapf_dingbats**, **antique_olive**, **arial**, **courier**, **garamond_antigua**, **letter_gothic**, **cg_omega**, **albertus**, **times_new_roman**, **clarendon**, **coronet**, **marigold**, **truetype_symbols**, or **wingdings**. <fontsize> is the font size in points. The point type selection can be the standard default set by specifying nopspoints, or the same set of point types found in the postscript terminal by specifying pspoints.

Note that built-in support of some of these options is printer device dependendent. For instance, all the fonts are supposedly supported by the HP Laserjet IV, but only a few (e.g. univers, stick) may be supported by the HP Laserjet III and the Designjet 750C. Also, color obviously won't work on the the laserjets since they are monochrome devices.

Defaults: landscape, noextended, color (6 pens), solid, univers, 12 point,

```
and nopspoints.
```

With **pcl5** international characters are handled by the printer; you just put the appropriate 8-bit character codes into the text strings. You don't need to bother with **set encoding**.

HPGL graphics can be imported by many software packages.

34.49.34 Hpljii

The **hpljii** terminal driver supports the HP Laserjet Series II printer. The **hpdj** driver supports the HP DeskJet 500 printer. These drivers allow a choice of resolutions.

Syntax:

```
set terminal hpljii | hpdj {<res>}
```

where **res** may be 75, 100, 150 or 300 dots per inch; the default is 75. Rasterization at the higher resolutions may require a large amount of memory.

The **hp500c** terminal is similar to **hpdj**; **hp500c** additionally supports color and compression.

34.49.35 Hppj

The **hppj** terminal driver supports the HP PaintJet and HP3630 printers. The only option is the choice of font.

Syntax:

```
set terminal hppj {FNT5X9 | FNT9X17 | FNT13X25}
```

with the middle-sized font (FNT9X17) being the default.

34.49.36 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.

34.49.37 Iris4d

The **iris4d** terminal driver supports Silicon Graphics IRIS 4D computers. Its only option is 8- or 24-bit color depth. The default is 8.

Syntax:

```
set terminal iris4d {8 | 24}
```

The color depth is not really a choice – the value appropriate for the hardware should be selected.

When using 24-bit mode, the colors can be directly specified via the file .gnuplot_iris4d that is searched in the current directory and then in the home directory specified by the HOME environment variable. This file holds RGB values for the background, border, labels and nine plotting colors, in that order. For example, here is a file containing the default colors:

```
85   85   85     Background   (dark gray)
0    0    0      Boundary     (black)
170  0    170    Labeling     (magenta)
85   255  255    Plot Color 1 (light cyan)
170  0    0      Plot Color 2 (red)
0    170  0      Plot Color 3 (green)
255  85   255    Plot Color 4 (light magenta)
255  255  85     Plot Color 5 (yellow)
255  85   85     Plot Color 6 (light red)
85   255  85     Plot Color 7 (light green)
0    170  170    Plot Color 8 (cyan)
170  170  0      Plot Color 9 (brown)
```

This file must have exactly 12 lines of RGB triples. No empty lines are allowed, and anything after the third number on a line is ignored.

34.49.38 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.

34.49.39 Latex

The **latex** and **emtex** drivers allow two options.

Syntax:

```
set terminal latex | emtex {courier | roman | default} {<fontsize>}
```

fontsize may be any size you specify. The default is for the plot to inherit its font setting from the embedding document.

Unless your driver is capable of building fonts at any size (e.g. dvips), stick to the standard 10, 11 and 12 point sizes.

METAFONT users beware: METAFONT does not like odd sizes.

All drivers for LaTeX offer a special way of controlling text positioning: If any text string begins with '{', you also need to include a '}' at the end of the text, and the whole text will be centered both horizontally and vertically. If the text string begins with '[', you need to follow this with a position specification (up to two out of t,b,l,r), ']{', the text itself, and finally '}'. The text itself may be anything LaTeX can typeset as an LR-box. '\rule{}{}'s may help for best positioning.

Points, among other things, are drawn using the LaTeX commands "\Diamond" and "\Box". These commands no longer belong to the LaTeX2e core; they are included in the latexsym package, which is part of the base distribution and thus part of any LaTeX implementation. Please do not forget to use this package.

Points are drawn with the LaTex commands \Diamond and \Box. These commands do no longer belong to the LaTeX2e core, but are included in the latexsym-package in the base distribution, and are hence part of all LaTeX implementations. Please do not forget to use this package.

Examples: About label positioning: Use gnuplot defaults (mostly sensible, but sometimes not really best):

```
set title '\LaTeX\ -- $ \gamma $'
```

Force centering both horizontally and vertically:

```
set label '{\LaTeX\ -- $ \gamma $}' at 0,0
```

Specify own positioning (top here):

```
set xlabel '[t]{\LaTeX\ -- $ \gamma $}'
```

The other label – account for long ticlabels:

```
set ylabel '[r]{\LaTeX\ -- $ \gamma $\rule{7mm}{0pt}'
```

34.49.40 Linux

The **linux** driver has no additional options to specify. It looks at the environment variable GSVGAMODE for the default mode; if not set, it uses 1024x768x256 as default mode or, if that is not possible, 640x480x16 (standard VGA).

34.49.41 Macintosh

Several options may be set in the 'macintosh' driver.

Syntax:

```
set terminal macintosh {singlewin | multiwin} {vertical | novertical}
                       {size <width>, <height> | default}
```

'singlewin' limits the output to a single window and is useful for animations. 'multiwin' allows multiple windows. 'vertical' is only valid under the gx option. With this option, rotated text

```
   be drawn vertically. novertical turns this option off.
size <width>, <height> overrides the graph size set in the preferences
   dialog until it is cleared with either 'set term mac size default'
   or 'set term mac default'.

'set term mac size default' sets the window size settings to those set in
   the preferences dialog.
```

```
'set term mac default' sets all options to their default values.
   Default values: nogx, multiwin, novertical.

If you generate graphs under the multiwin option and then switch to singlewin,
the next plot command will cause one more window to be created. This new
window will be reused as long as singlewin is in effect. If you switch back
to multiwin, generate some graphs, and then switch to singlewin again, the
orginal 'singlewin' window will be resused if it is still open. Otherwise
a new 'singlewin' window will be created. The 'singlewin' window is not numbered.
```

34.49.42 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.

34.49.42.1 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.

34.49.43 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.

The **mp** terminal is selected with a command of the form

```
set term mp {color} {solid} {notex} {mag <magsize>} {"<name>"} {<size>}
```

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.

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.

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 processess 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.

34.49.43.1 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.

34.49.44 Mgr

The **mgr** terminal driver supports the Mgr Window system. It has no options.

34.49.45 Mif

The **mif** terminal driver produces Frame Maker MIF format version 3.00. It plots in MIF Frames with the size 15*10 cm, and plot primitives with the same pen will be grouped in the same MIF group. Plot primitives in a **gnuplot** page will be plotted in a MIF Frame, and several MIF Frames are collected in one large MIF Frame. The MIF font used for text is "Times".

Several options may be set in the MIF 3.00 driver.

Syntax:

```
set terminal mif {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
```

34.49.46 Mtos

The **mtos** terminal has no options. It sends data via a pipe to an external program called GPCLIENT. It runs under MULTITOS, Magic 3.x, MagicMAC. and MiNT. If you cannot find GPCLIENT, than mail to dirk@lstm.uni-erlangen.de.

34.49.47 Next

Several options may be set in the next driver.

Syntax:

```
set terminal next {<mode>} {<type> } {<color>} {<dashed>}
          {"<fontname>"} {<fontsize>} title {"<newtitle>"}
```

where <mode> is **default**, which sets all options to their defaults; <type> is either **new** or **old**, where **old** invokes the old single window; <color> is either **color** or **monochrome**; <dashed> is either **solid** or **dashed**; "<fontname>" is the name of a valid PostScript font; <fontsize> is the size of the font in PostScript points; and <title> is the title for the GnuTerm window. Defaults are **new**, **monochrome**, **dashed**, "Helvetica", 14pt.

Examples:

```
set term next default
set term next 22
set term next color "Times-Roman" 14
set term next color "Helvetica" 12 title "MyPlot"
set term next old
```

Pointsizes may be changed with **set linestyle**.

34.49.48 Next

Several options may be set in the next driver.

Syntax:

```
set terminal next {<mode>} {<type> } {<color>} {<dashed>}
          {"<fontname>"} {<fontsize>} title {"<newtitle>"}
```

where <mode> is **default**, which sets all options to their defaults; <type> is either **new** or **old**, where **old** invokes the old single window; <color> is either **color** or **monochrome**; <dashed> is either **solid** or **dashed**; "<fontname>" is the name of a valid PostScript font; <fontsize> is the size of the font in PostScript points; and <title> is the title for the GnuTerm window. Defaults are **new**, **monochrome**, **dashed**, "Helvetica", 14pt.

Examples:

```
set term next default
set term next 22
set term next color "Times-Roman" 14
set term next color "Helvetica" 12 title "MyPlot"
set term next old
```

Pointsizes may be changed with **set linestyle**.

34.49.49 Pbm

Several options may be set in the **pbm** terminal — the driver for PBMplus.

Syntax:

```
set terminal pbm {<fontsize>} {<mode>}
```

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; this may be changed by **set size**.

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 Jef Poskanzer's excellent PBMPLUS package, which provides programs to convert the above PBMPLUS formats to GIF, TIFF, MacPaint, Macintosh PICT, PCX, X11 bitmap and many others. PBMPLUS may be obtained from ftp.x.org. The relevant files have names that begin with "netpbm-1mar1994.p1"; they reside in /contrib/utilities. The package can probably also be obtained from one of the many sites that mirrors ftp.x.org.

Examples:

```
set terminal pbm small monochrome             # defaults
set size 2,2; set terminal pbm color medium
```

34.49.50 Dospc

The **dospc** terminal driver supports PCs with arbitrary graphics boards, which will be automatically detected. It should be used only if you are not using the gcc or Zortec C/C++ compilers.

34.49.51 Pdf

This terminal produces files in the Adobe Portable Document Format (PDF), useable for printing or display with tools like Acrobat Reader

Syntax:

```
set terminal pdf {fname "<font>"} {fsize <fontsize>}
```

where <font> is the name of the default font to use (default Helvetica) and <fontsize> is the font size (in points, default 12)

34.49.52 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, copying to the clipboard and some line type and color adjustments. The **multiplot** option is supported.

Syntax:

```
set terminal pm {server {n}} {persist} {widelines} {enhanced} {"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 using the same syntax as the **enhanced postscript** option (see **set terminal postscript enhanced** for details). Font names for the basic PostScript fonts may be abbreviated to single letters.

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.

Linewidths may be changed with **set linestyle**.

34.49.53 Png

The **png** terminal driver supports Portable Network Graphics. To compile it, you will need the third-party libraries "libpng" and "zlib"; both are available at http://www.cdrom.com/pub/png/. **png** has four options.

By default, the **png** terminal driver uses a shared Web-friendy palette.

Syntax:

```
set terminal png {small | medium | large}
                 {transparent | notransparent}
                 {monochrome | gray | color}
                 {<color0> <color1> <color2> ...}
```

transparent instructs the driver to generate transparent PNGs. The first color will be the transparent one.

The defaults are small (fontsize) and color. Default size of the output is 640*480 pixel.

Each color must be of the form 'xrrggbb', where x is the literal character 'x' and 'rrggbb' are the red, green and blue components in hex. For example, 'x00ff00' is green. The background color is set first, then the border colors, then the X & Y axis colors, then the plotting colors. The maximum number of colors that can be set is currently 99.

34.49.54 Postscript

Several options may be set in the **postscript** driver.

Syntax:

```
set terminal postscript {<mode>} {enhanced | noenhanced}
                        {color | monochrome} {solid | dashed}
                        {<duplexing>}
                        {"<fontname>"} {<fontsize>}
```

where <mode> is **landscape**, **portrait**, **eps** or **default**; **solid** draws all plots with solid lines, overriding any dashed patterns; <duplexing> is **defaultplex**, **simplex** or **duplex** ("duplexing" in PostScript is the ability of the printer to print on both sides of the same page — don't set this if your printer can't do it); **enhanced** activates the "enhanced PostScript" features (subscripts, superscripts and mixed fonts); "**<fontname>**" is the name of a valid PostScript font; and **<fontsize>** is the size of the font in PostScript points.

default mode sets all options to their defaults: **landscape**, **monochrome**, **dashed**, **defaultplex**, **noenhanced**, "Helvetica" and 14pt.

```
Default size of a PostScript plot is 10 inches wide and 7 inches high.
```

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.

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
```

Linewidths and pointsizes may be changed with **set linestyle**.

The **postscript** driver supports about 70 distinct pointtypes, selectable through the **pointtype** option on **plot** and **set linestyle**.

Several possibly useful files about **gnuplot**'s PostScript are included in the /docs/ps 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) and "ps_file.doc" (a text file that contains a discussion of the organization of a PostScript file written by **gnuplot**).

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" section for some hints.

34.49.54.1 Enhanced postscript

Enhanced Text Control Codes		
Control	Examples	Explanation
`^`	`a^x`	superscript
`_`	`a_x`	subscript
`@`	`@x or a@^b_c`	phantom box (occupies no width)
`&`	`&{space}`	inserts space of specified length

Braces can be used to place multiple-character text where a single character is expected (e.g., 2^{10}). To change the font and/or size, use the full form: {/[fontname][=fontsize | *fontscale] text}. Thus {/Symbol=20 G} is a 20-point GAMMA) and {/*0.75 K} is a K at three-quarters of whatever fontsize is currently in effect. (The '/' character MUST be the first character after the '{'.)

If the encoding vector has been changed by **set encoding**, the default encoding vector can be used instead by following the slash with a dash. This is unnecessary if you use the Symbol font, however — since /Symbol uses its own encoding vector, **gnuplot** will not apply any other encoding vector to it.

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. (To do the latter, it is much better to use **set encoding iso_8859_1** to change to the ISO Latin-1 encoding vector, which contains a large variety of letters with accents or other diacritical marks.) 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'.
```

You can access special symbols numerically by specifying \character-code (in octal), e.g., {/Symbol \245} is the symbol for infinity.

You can escape control characters using \, e.g., \\, \{, and so on.

But be aware 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.

Examples (these are hard to describe in words — try them!):

```
set xlabel 'Time (10^6 {/Symbol m}s)'
set title '{/Symbol=18 \362@_{/=9.6 0}^{/=12 x}} \
           {/Helvetica e^{-{/Symbol m}^2/2} d}{/Symbol m}'
```

The file "ps_guide.ps" in the /docs/ps subdirectory of the **gnuplot** source distribution contains more examples of the enhanced syntax.

34.49.54.2 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.

34.49.55 Pslatex and pstex

The **pslatex** and **pstex** drivers generate output for further processing by LaTeX and TeX, respectively. Figures generated by **pstex** can be included in any plain-based format (including LaTeX).

Syntax:

```
set terminal pslatex | |pstex {<color>} {<dashed>} {<rotate>}
                              {auxfile} {<font_size>}
```

<color> is either **color** or **monochrome**. <rotate> is either **rotate** or **norotate** and determines if 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. Remember to close the file before leaving **gnuplot**.

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:

```
set term pslatex monochrome dashed 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 linestyle**.

34.49.56 Pstricks

The **pstricks** driver is intended for use with the "pstricks.sty" macro package for LaTeX. It is an alternative to the **eepic** and **latex** drivers. You need "pstricks.sty", and, of course, a printer that understands PostScript, or a converter such as Ghostscript.

PSTricks is available via anonymous ftp from the /pub directory at Princeton.EDU. This driver definitely does not come close to using the full capability of the PSTricks package.

Syntax:

```
set terminal pstricks {hacktext | nohacktext} {unit | nounit}
```

The first option invokes an ugly hack that gives nicer numbers; the second has to do with plot scaling. The defaults are **hacktext** and **nounit**.

34.49.57 Qms

The **qms** terminal driver supports the QMS/QUIC Laser printer, the Talaris 1200 and others. It has no options.

34.49.58 Regis

The **regis** terminal device generates output in the REGIS graphics language. It has the option of using 4 (the default) or 16 colors.

Syntax:

```
set terminal regis {4 | 16}
```

34.49.59 Rgip

The **rgip** and **uniplex** terminal drivers support RGIP metafiles. They can combine several graphs on a single page, but only one page is allowed in a given output file.

Syntax:

```
set terminal rgip | uniplex {portrait | landscape}
                            {[<horiz>,<vert>]} {<fontsize>}
```

permissible values for the font size are in the range 1–8, with the default being 1. The default layout is landscape. Graphs are placed on the page in a **horiz**x**vert** grid, which defaults to [1,1].

Example:

```
set terminal uniplex portrait [2,3]
```

puts six graphs on a page in three rows of two in portrait orientation.

34.49.60 Sun

The **sun** terminal driver supports the SunView window system. It has no options.

34.49.61 Svg

This terminal produces files in the W3C Scalable Vector Graphics format.

Syntax:

```
set terminal svg {size <x> <y>}
                 {fname "<font>"} {fsize <fontsize>}
```

where <x> and <y> are the size of the SVG plot to generate, <font> is the name of the default font to use (default Arial) and <fontsize> is the font size (in points, default 12)

34.49.62 Tek410x

The **tek410x** terminal driver supports the 410x and 420x family of Tektronix terminals. It has no options.

34.49.63 Table

Instead of producing a graph, the **table** terminal prints out the points on which a graph would be based, i.e., the results of processing the **plot** or **splot** command, in a multicolumn ASCII table of X Y {Z} R values. The character R takes on one of three values: "i" if the point is in the active range, "o" if it is out-of-range, or "u" if it is undefined. The data format is determined by the format of the axis labels (see **set format**), and the columns are separated by single spaces.

For those times when you want the numbers, you can display them on the screen or save them to a file. This can be useful if you want to generate contours and then save them for further use, perhaps for plotting with **plot**; see **set contour** for an example. The same method can be used to save interpolated data (see **set samples** and **set dgrid3d**).

34.49.64 Tek40

This family of terminal drivers supports a variety of VT-like terminals. **tek40xx** supports Tektronix 4010 and others as well as most TEK emulators; **vttek** supports VT-like tek40xx terminal emulators; **kc-tek40xx** supports MS-DOS Kermit Tek4010 terminal emulators in color: **km-tek40xx** supports them in monochrome; **selanar** supports Selanar graphics; and **bitgraph** supports BBN Bitgraph terminals. None have any options.

34.49.65 Texdraw

The **texdraw** terminal driver supports the LaTeX texdraw environment. It is intended for use with "texdraw.sty" and "texdraw.tex" in the texdraw package.

It has no options.

34.49.66 Tgif

Tgif is an X11-based drawing tool — it has nothing to do with GIF.

The **tgif** driver supports different pointsizes (with **set pointsize**), different label fonts and font sizes (e.g. **set label "Hallo" at x,y font "Helvetica,34"**) and multiple graphs on the page. The proportions of the axes are not changed.

Syntax:

```
set terminal tgif {portrait | landscape} {<[x,y]>}
                  {solid | dashed}
                  {"<fontname>"} {<fontsize>}
```

where <[x,y]> specifies the number of graphs in the x and y directions on the page, "<fontname>" is the name of a valid PostScript font, and <fontsize> specifies the size of the PostScript font. Defaults are **portrait**, **[1,1]**, **dashed**, **"Helvetica"**, and **18**.

The **solid** option is usually prefered if lines are colored, as they often are in the editor. Hardcopy will be black-and-white, so **dashed** should be chosen for that.

Multiplot is implemented in two different ways.

The first multiplot implementation is the standard gnuplot multiplot feature:

```
set terminal tgif
set output "file.obj"
set multiplot
set origin x01,y01
set size  xs,ys
plot ...
     ...
set origin x02,y02
plot ...
set nomultiplot
```

See **set multiplot** 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
```

34.49.67 Tkcanvas

This terminal driver generates Tk canvas widget commands based on Tcl/Tk (default) or Perl. To use it, rebuild **gnuplot** (after uncommenting or inserting the appropriate line in "term.h"), then

```
gnuplot> set term tkcanvas {perltk} {interactive}
gnuplot> set output 'plot.file'
```

After invoking "wish", execute the following sequence of Tcl/Tk commands:

```
% source plot.file
% canvas .c
% pack .c
% gnuplot .c
```

Or, for Perl/Tk use a program like this:

```
use Tk;
my $top = MainWindow->new;
my $c = $top->Canvas->pack;
my $gnuplot = do "plot.pl";
$gnuplot->($c);
MainLoop;
```

The code generated by **gnuplot** creates a procedure called "gnuplot" that takes the name of a canvas as its argument. When the procedure is called, it clears the canvas, finds the size of the canvas and draws the plot in it, scaled to fit.

For 2-dimensional plotting (**plot**) two additional procedures are defined: "gnuplot_plotarea" will return a list containing the borders of the plotting area "xleft, xright, ytop, ybot" in canvas screen coordinates, while the ranges of the two axes "x1min, x1max, y1min, y1max, x2min, x2max, y2min, y2max" in plot coordinates can be obtained calling "gnuplot_axisranges". If the "interactive" option is specified, mouse clicking on a line segment will print the coordinates of its midpoint to stdout. Advanced actions can happen instead if the user supplies a procedure named "user_gnuplot_coordinates", which takes the following arguments: "win id x1s y1s x2s y2s x1e y1e x2e y2e x1m y1m x2m y2m", the name of the canvas and the id of the line segment followed by the coordinates of its start and end point in the two possible axis ranges; the coordinates of the midpoint are only filled for logarithmic axes.

The current version of **tkcanvas** supports neither **multiplot** nor **replot**.

34.49.68 Tpic

The **tpic** terminal driver supports the LaTeX picture environment with tpic \specials. It is an alternative to the **latex** and **eepic** terminal drivers. Options are the point size, line width, and dot-dash interval.

Syntax:

```
set terminal tpic <pointsize> <linewidth> <interval>
```

where **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}'
```

34.49.69 Unixpc

The **unixpc** terminal driver supports AT&T 3b1 and AT&T 7300 Unix PC. It has no options.

34.49.70 Unixplot

The **unixplot** terminal driver generates output in the Unix "plot" graphics language. It has no options.

This terminal cannot be compiled if the GNU version of plot is to be used; in that case, use the **gnugraph** terminal instead.

34.49.71 Vx384

The **vx384** terminal driver supports the Vectrix 384 and Tandy color printers. It has no options.

34.49.72 VWS

The **VWS** terminal driver supports the VAX Windowing System. It has no options. It will sense the display type (monochrome, gray scale, or color.) All line styles are plotted as solid lines.

34.49.73 Windows

Three options may be set in the **windows** terminal driver.

Syntax:

```
set terminal windows {<color>} {"<fontname>"} {<fontsize>}
```

where <**color**> is either **color** or **monochrome**, "<**fontname**>" is the name of a valid Windows font, and <**fontsize**> is the size of the font in points.

Other options may be set with the graph-menu, the initialization file, and **set linestyle**.

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). It will also not show the text-window at all, in this mode, only the plot. By giving the optional argument **/noend** or **-noend**, you can disable this behaviour.

34.49.73.1 Graph-menu The **gnuplot graph** window has the following options on a pop-up menu accessed by pressing the right mouse button or selecting **Options** from the system menu:

Bring to Top when checked brings the graph window to the top after every plot.

Color when checked enables color linestyles. When unchecked it forces monochrome linestyles.

Copy to Clipboard copies a bitmap and a Metafile picture.

Background... sets the window background color.

Choose Font... selects the font used in the graphics window.

Line Styles... allows customization of the line colors and styles.

Print... prints the graphics windows using a Windows printer driver and allows selection of the printer and scaling of the output. The output produced by **Print** is not as good as that from **gnuplot**'s own printer drivers.

Update wgnuplot.ini saves the current window locations, window sizes, text window font, text window font size, graph window font, graph window font size, background color and linestyles to the initialization file **WGNUPLOT.INI**.

34.49.73.2 Printing In order of preference, graphs may be be printed in the following ways.

1. Use the **gnuplot** command **set terminal** to select a printer and **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.

34.49.73.3 Text-menu The **gnuplot text** window has the following options on a pop-up menu accessed by pressing the right mouse button or selecting **Options** from the system menu:

Copy to Clipboard copies marked text to the clipboard.

Paste copies text from the clipboard as if typed by the user.

Choose Font... selects the font used in the text window.

System Colors when selected makes the text window honor the System Colors set using the Control Panel. When unselected, text is black or blue on a white background.

Update wgnuplot.ini saves the current text window location, text window size, text window font and text window font size to the initialisation file **WGNUPLOT.INI**.

MENU BAR

If the menu file **WGNUPLOT.MNU** is found in the same directory as WGNUPLOT.EXE, then the menu specified in **WGNUPLOT.MNU** will be loaded. Menu commands:

[Menu] starts a new menu with the name on the following line.

[EndMenu] ends the current menu.

[–] inserts a horizontal menu separator.

[|] inserts a vertical menu separator.

[Button] puts the next macro on a push button instead of a menu.

Macros take two lines with the macro name (menu entry) on the first line and the macro on the second line. Leading spaces are ignored. Macro commands:

[INPUT] — Input string with prompt terminated by [EOS] or {ENTER}

[EOS] — End Of String terminator. Generates no output.

[OPEN] — Get name of file to open from list box, with title of list box terminated by [EOS], followed by default filename terminated by [EOS] or {ENTER}. This uses COMMDLG.DLL from Windows 3.1.

[SAVE] — Get name of file to save. Similar to [OPEN]

Macro character substitutions:

{ENTER} — Carriage Return '\r'

{TAB} — Tab '\011'

{ESC} — Escape '\033'

{^A} — '\001'

...

{^_} — '\031'

Macros are limited to 256 characters after expansion.

34.49.73.4 Wgnuplot.ini Windows **gnuplot** will read some of its options from the [**WGNUPLOT**] section of **WGNUPLOT.INI** in the Windows directory. A sample **WGNUPLOT.INI** file:

```
[WGNUPLOT]
TextOrigin=0 0
TextSize=640 150
TextFont=Terminal,9
GraphOrigin=0 150
GraphSize=640 330
GraphFont=Arial,10
GraphColor=1
GraphToTop=1
```

```
GraphBackground=255 255 255
Border=0 0 0 0 0
Axis=192 192 192 2 2
Line1=0 0 255 0 0
Line2=0 255 0 0 1
Line3=255 0 0 0 2
Line4=255 0 255 0 3
Line5=0 0 128 0 4
```

The **GraphFont** entry specifies the font name and size in points. The five numbers given in the **Border**, **Axis** and **Line** entries are the **Red** intensity (0–255), **Green** intensity, **Blue** intensity, **Color Linestyle** and **Mono Linestyle**. **Linestyles** are 0=SOLID, 1=DASH, 2=DOT, 3=DASHDOT, 4=DASHDOTDOT. In the sample **WGNUPLOT.INI** file above, Line 2 is a green solid line in color mode, or a dashed line in monochrome mode. The default line width is 1 pixel. If **Linestyle** is negative, it specifies the width of a SOLID line in pixels. Line1 and any linestyle used with the **points** style must be SOLID with unit width.

34.49.73.5 Windows3.0 Windows 3.1 is preferred, but WGNUPLOT will run under Windows 3.0 with the following restrictions: **1.** COMMDLG.DLL and SHELL.DLL (available with Windows 3.1 or Borland C++ 3.1) must be in the windows directory.

2. WGNUPLOT.HLP produced by Borland C++ 3.1 is in Windows 3.1 format. You need to use the WINHELP.EXE supplied with Borland C++ 3.1.

3. It will not run in real mode due to lack of memory.

4. TrueType fonts are not available in the graph window.

5. Drag-drop does not work.

34.49.74 X11

gnuplot provides the **x11** terminal type for use with X servers. This terminal type is set automatically at startup if the **DISPLAY** environment variable is set, if the **TERM** environment variable is set to **xterm**, or if the **-display** command line option is used.

Syntax:

```
set terminal x11 {reset} {<n>}
```

Multiple plot windows are supported: **set terminal x11 <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 **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.

34.49.74.1 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.
'-tvtwm'	requests that geometry specifications for position of the window be made relative to the currently displayed portion of the virtual root.
'-raise'	raise plot window after each plot.
'-noraise'	do 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 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.

34.49.74.2 Monochome_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.

34.49.74.3 Color_resources For color displays, **gnuplot** honors the following resources (shown here with their default values) or the greyscale resources. The values may be color names as listed in the X11 rgb.txt file on your system, hexadecimal RGB color specifications (see X11 documentation), or a color name followed by a comma and an **intensity** value from 0 to 1. For example, **blue, 0.5** means a half intensity blue.

gnuplot*background: white
gnuplot*textColor: black
gnuplot*borderColor: black
gnuplot*axisColor: black
gnuplot*line1Color: red
gnuplot*line2Color: green
gnuplot*line3Color: blue
gnuplot*line4Color: magenta
gnuplot*line5Color: cyan
gnuplot*line6Color: sienna
gnuplot*line7Color: orange
gnuplot*line8Color: coral

The command-line syntax for these is, for example,

Example:

```
gnuplot -background coral
```

34.49.74.4 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
```

34.49.74.5 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
```

34.49.75 Xlib

The **xlib** terminal driver supports the X11 Windows System. It generates gnulib_x11 commands. **set term x11** behaves similarly to **set terminal xlib; set output "|gnuplot_x11"**. **xlib** has no options, but see **x11**.

34.50 Tics

The **set tics** command can be used to change the tics to be drawn outwards.

Syntax:

```
set tics {<direction>}
show tics
```

where <direction> may be **in** (the default) or **out**.

See also **set xtics** for more control of major (labelled) tic marks and **set mxtics** for control of minor tic marks.

34.51 Ticslevel

Using **splot**, one can adjust the relative height of the vertical (Z) axis using **set ticslevel**. The numeric argument provided specifies the location of the bottom of the scale (as a fraction of the z-range) above the xy-plane. The default value is 0.5. Negative values are permitted, but tic labels on the three axes may overlap.

To place the xy-plane at a position 'pos' on the z-axis, **ticslevel** should be set equal to (pos - zmin) / (zmin - zmax).

Syntax:

```
set ticslevel {<level>}
show tics
```

See also **set view**.

34.52 Ticscale

The size of the tic marks can be adjusted with **set ticscale**.

Syntax:

```
set ticscale {<major> {<minor>}}
show tics
```

If <minor> is not specified, it is 0.5*<major>. The default size is 1.0 for major tics and 0.5 for minor tics. Note that it is possible to have the tic marks pointing outward by specifying a negative size.

34.53 Timestamp

The command **set timestamp** places the time and date of the plot in the left margin.

Syntax:

```
set timestamp {"<format>"} {top|bottom} {{no}rotate}
              {<xoff>}{,<yoff>} {"<font>"}
set notimestamp
show timestamp
```

The format string allows you to choose the format used to write the date and time. Its default value is what asctime() uses: "%a %b %d %H:%M:%S %Y" (weekday, month name, day of the month, hours, minutes, seconds, four-digit year). With **top** or **bottom** you can place the timestamp at the top or bottom of the left margin (default: bottom). **rotate** lets you write the timestamp vertically, if your terminal supports vertical text. The constants <xoff> and <off> are offsets from the default position given in character screen coordinates. <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" 80,-2 "Helvetica"
```

See **set timefmt** for more information about time format strings.

34.54 Timefmt

This command applies to timeseries where data are composed of dates/times. It has no meaning unless the command **set xdata time** is given also.

Syntax:

```
set timefmt "<format string>"
show timefmt
```

The string argument tells **gnuplot** how to read timedata from the datafile. The valid formats are:

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`	second, 0–60
`%b`	three-character abbreviation of the name of the month
`%B`	name of the month

Any character is allowed in the string, but must match exactly. \t (tab) is recognized. Backslash-octals (\nnn) are converted to char. If there is no separating character between the time/date elements, then %d, %m, %y, %H, %M and %S read two digits each, %Y reads four digits and %j reads three digits. %b requires three characters, and %B requires as many as it needs.

Spaces are treated slightly differently. A space in the string stands for zero or more whitespace characters in the file. That is, "%H %M" can be used to read "1220" and "12 20" as well as "12 20".

Each set of non-blank characters in the timedata counts as one column in the **using n:n** specification. Thus **11:11 25/12/76 21.0** consists of three columns. To avoid confusion, **gnuplot** requires that you provide a complete **using** specification if your file contains timedata.

Since **gnuplot** cannot read non-numerical text, if the date format includes the day or month in words, the format string must exclude this text. But it can still be printed with the "%a", "%A", "%b", or "%B" specifier: see **set format** for more details about these and other options for printing timedata. (**gnuplot** will determine the proper month and weekday from the numerical values.)

See also **set xdata** and **Time/date** for more information.

Example:

```
set timefmt "%d/%m/%Y\t%H:%M"
```

tells **gnuplot** to read date and time separated by tab. (But look closely at your data — what began as a tab may have been converted to spaces somewhere along the line; the format string must match what is actually in the file.)

34.55 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>"} {<xoff>}{,<yoff>} {"<font>,{<size>}"}
show title
```

Specifying constants <xoff> or <yoff> as optional offsets for the title will move the title <xoff> or <yoff> character screen coordinates (not graph coordinates). For example, **"set title ,-1"** will change only the y offset of the title, moving the title down by roughly the height of one character.

<font> is used to specify the font with which the title is to be written; the units of the font <size> depend upon which terminal is used.

set title with no parameters clears the title.

See **syntax** for details about the processing of backslash sequences and the distinction between single- and double-quotes.

34.56 Tmargin

The command **set tmargin** sets the size of the top margin. Please see **set margin** for details.

34.57 Trange

The **set trange** command sets the parametric range used to compute x and y values when in parametric or polar modes. Please see **set xrange** for details.

34.58 Urange

The **set urange** and **set vrange** commands set the parametric ranges used to compute x, y, and z values when in **splot** parametric mode. Please see **set xrange** for details.

34.59 Variables

The **show variables** command lists all user-defined variables and their values.

Syntax:

```
show variables
```

34.60 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 info-gnuplot 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.

34.61 View

The **set view** command sets the viewing angle for **splot**s. It controls how the 3-d coordinates of the plot are mapped into the 2-d screen space. It provides controls for both rotation and scaling of the plotted data, but supports orthographic projections only.

Syntax:

```
set view <rot_x> {,{<rot_z>}{,{<scale>}{,<scale_z>}}}
show view
```

where <rot_x> and <rot_z> control the rotation angles (in degrees) in a virtual 3-d coordinate system aligned with the screen such that initially (that is, before the rotations are performed) the screen horizontal axis is x, screen vertical axis is y, and the axis perpendicular to the screen is z. The first rotation applied is <rot_x> around the x axis. The second rotation applied is <rot_z> around the new z axis.

<rot_x> is bounded to the [0:180] range with a default of 60 degrees, while <rot_z> is bounded to the [0:360] range with a default of 30 degrees. <scale> controls the scaling of the entire **splot**, while <scale_z> scales the z axis only. Both scales default to 1.0.

Examples:

```
set view 60, 30, 1, 1
set view ,,0.5
```

The first sets all the four default values. The second changes only scale, to 0.5.

See also **set ticslevel**.

34.62 Vrange

The **set urange** and **set vrange** commands set the parametric ranges used to compute x, y, and z values when in **splot** parametric mode. Please see **set xrange** for details.

34.63 X2data

The **set x2data** command sets data on the x2 (top) axis to timeseries (dates/times). Please see **set xdata**.

34.64 X2dtics

The **set x2dtics** command changes tics on the x2 (top) axis to days of the week. Please see **set xdtics** for details.

34.65 X2label

The **set x2label** command sets the label for the x2 (top) axis. Please see **set xlabel**.

34.66 X2mtics

The **set x2mtics** command changes tics on the x2 (top) axis to months of the year. Please see **set xmtics** for details.

34.67 X2range

The **set x2range** command sets the horizontal range that will be displayed on the x2 (top) axis. Please see **set xrange** for details.

34.68 X2tics

The **set x2tics** command controls major (labelled) tics on the x2 (top) axis. Please see **set xtics** for details.

34.69 X2zeroaxis

The **set x2zeroaxis** command draws a line at the origin of the x2 (top) axis (y2 = 0). For details, please see **set zeroaxis**.

34.70 Xdata

This command sets the datatype on the x axis to time/date. A similar command does the same thing for each of the other axes.

Syntax:

```
set xdata {time}
show xdata
```

The same syntax applies to **ydata**, **zdata**, **x2data** and **y2data**.

The **time** option signals that the datatype is indeed time/date. If the option is not specified, the datatype reverts to normal.

See **set timefmt** to tell **gnuplot** how to read date or time data. The time/date is converted to seconds from start of the century. There is currently only one timefmt, which implies that all the time/date columns must confirm to this format. Specification of ranges should be supplied as quoted strings according to this format to avoid interpretation of the time/date as an expression.

The function 'strftime' (type "man strftime" on unix to look it up) is used to print tic-mark labels. **gnuplot** tries to figure out a reasonable format for this unless the **set format x "string"** has supplied something that does not look like a decimal format (more than one '%' or neither %f nor %g).

See also **Time/date** for more information.

34.71 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
set noxdtics
show xdtics
```

The same syntax applies to **ydtics**, **zdtics**, **x2dtics** and **y2dtics**.

See also the **set format** command.

34.72 Xlabel

The **set xlabel** command sets the x axis label. Similar commands set labels on the other axes.

Syntax:

```
set xlabel {"<label>"} {<xoff>}{,<yoff>} {"<font>{,<size>}"}
show xlabel
```

The same syntax applies to **x2label**, **ylabel**, **y2label** and **zlabel**.

Specifying the constants <xoff> or <yoff> as optional offsets for a label will move it <xoff> or <yoff> character widths or heights. For example, **"set xlabel -1"** will change only the x offset of the xlabel, 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.

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 axis.

ylabel: The position of the y-axis label depends on the terminal, and can be one of the following three positions:

1. Horizontal text flushed left at the top left of the plot. Terminals that cannot rotate text will probably use this method. If **set x2tics** is also in use, the ylabel may overwrite the left-most x2tic label. This may be remedied by adjusting the ylabel position or the left margin.

2. Vertical text centered vertically at the left of the plot. Terminals that can rotate text will probably use this method.

3. Horizontal text centered vertically at the left of the plot. The EEPIC, LaTeX and TPIC drivers use this method. The user must insert line breaks using \\ to prevent the ylabel from overwriting the plot. To produce a vertical row of characters, add \\ between every printing character (but this is ugly).

zlabel: The z-axis label is centered along the z axis and placed in the space above the grid level.

y2label: The y2-axis label is placed to the right of the y2 axis. The position is terminal-dependent in the same manner as is the y-axis label.

x2label: The x2-axis label is placed above the top axis but below the plot title. It is also possible to create an x2-axis label by using new-line characters to make a multi-line plot title, e.g.,

```
set title "This is the title\n\nThis is the x2label"
```

Note that double quotes must be used. The same font will be used for both lines, of course.

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 **set syntax** for further information about backslash processing and the difference between single- and double-quoted strings.

34.73 Xmtics

The **set xmtics** commands 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 **set noxmtics**. Similar commands perform the same duties for the other axes.

Syntax:

```
set xmtics
```

```
set noxmtics
show xmtics
```

The same syntax applies to **x2mtics**, **ymtics**, **y2mtics**, and **zmtics**.

See also the **set format** command.

34.74 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} }
           | 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. Any value omitted will not be changed.

The same syntax applies to **yrange**, **zrange**, **x2range**, **y2range**, **rrange**, **trange**, **urange** and **vrange**.

The **reverse** option reverses the direction of the axis, e.g., **set xrange [0:1] reverse** will produce an axis with 1 on the left and 0 on the right. This is identical to the axis produced by **set xrange [1:0]**, of course. **reverse** is intended primarily for use with **autoscale**.

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 2-d, **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 3-d, **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:

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 [*:]
```

34.75 Xtics

Fine control of the major (labelled) tics on the x axis is possible with the **set xtics** command. The tics may be turned off with the **set noxtics** 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} {{no}rotate}
          {  autofreq
           | <incr>
           | <start>, <incr> {,<end>}
           | ({"<label>"} <pos> {,{"<label>"} <pos>}...) }
set noxtics
show xtics
```

The same syntax applies to **ytics**, **ztics**, **x2tics** and **y2tics**.

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 can result in tic labels overwriting other text written in the margin.

mirror tells **gnuplot** to put unlabelled tics at the same positions on the opposite border. **nomirror** does what you think it does.

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.

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 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**.

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.

Positions of the tics are calculated automatically by default or if the **autofreq** option is given; otherwise they may be specified in either of two forms:

The implicit <start>, <incr>, <end> form specifies that a series of tics will be plotted on the axis between the values <start> and <end> with an increment of <incr>. If <end> is not given, it is assumed to be infinity. The increment may be negative. If neither <start> nor <end> is given, <start> is assumed to be negative infinity, <end> is assumed to be positive infinity, and the tics will be drawn at integral multiples of <step>. If the axis is logarithmic, the increment will be used as a multiplicative factor.

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 1,100,10e8
```

The explicit ("<label>" <pos>, ...) form allows arbitrary tic positions or non-numeric tic labels. A set of tics is a set of positions, each with its own optional label. Note that the label is a string enclosed by quotes. It may be a constant string, such as "hello", may contain formatting information for converting the position into its label, such as "%3f clients", or may be empty, "". See **set format** for more information. If no string is given, the default label (numerical) is used. In this form, the tics do not need to be listed in numerical order.

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)
```

In the second example, all tics are labelled. In the third, only the end tics are labelled.

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 labels is included in the **set xtic (**<label>**)** form.

Minor (unlabelled) tics can be added by the **set mxtics** command.

In case of timeseries data, position values must be given as quoted dates or times according to the format **timefmt**. If the <start>, <incr>, <end> form is used, <start> and <end> must be given according to **timefmt**, but <incr> must be in seconds. Times will be written out according to the format given on **set format**, however.

Examples:

```
set xdata time
set timefmt "%d/%m"
set format x "%b %d"
set xrange ["01/12":"06/12"]
set xtics "01/12", 172800, "05/12"

set xdata time
set timefmt "%d/%m"
set format x "%b %d"
set xrange ["01/12":"06/12"]
set xtics ("01/12", "" "03/12", "05/12")
```

Both of these will produce tics "Dec 1", "Dec 3", and "Dec 5", but in the second example the tic at "Dec 3" will be unlabelled.

34.76 Xzeroaxis

The **set xzeroaxis** command draws a line at y = 0. For details, please see **set zeroaxis**.

34.77 Y2data

The **set y2data** command sets y2 (right-hand) axis data to timeseries (dates/times). Please see **set xdata**.

34.78 Y2dtics

The **set y2dtics** command changes tics on the y2 (right-hand) axis to days of the week. Please see **set xdtics** for details.

34.79 Y2label

The **set y2dtics** command sets the label for the y2 (right-hand) axis. Please see **set xlabel**.

34.80 Y2mtics

The **set y2mtics** command changes tics on the y2 (right-hand) axis to months of the year. Please see **set xmtics** for details.

34.81 Y2range

The **set y2range** command sets the vertical range that will be displayed on the y2 (right-hand) axis. Please see **set xrange** for details.

34.82 Y2tics

The **set y2tics** command controls major (labelled) tics on the y2 (right-hand) axis. Please see **set xtics** for details.

34.83 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**.

34.84 Ydata

Sets y-axis data to timeseries (dates/times). Please see **set xdata**.

34.85 Ydtics

The **set ydtics** command changes tics on the y axis to days of the week. Please see **set xdtics** for details.

34.86 Ylabel

This command sets the label for the y axis. Please see **set xlabel**.

34.87 Ymtics

The **set ymtics** command changes tics on the y axis to months of the year. Please see **set xmtics** for details.

34.88 Yrange

The **set yrange** command sets the vertical range that will be displayed on the y axis. Please see **set xrange** for details.

34.89 Ytics

The **set ytics** command controls major (labelled) tics on the y axis. Please see **set xtics** for details.

34.90 Yzeroaxis

The **set yzeroaxis** command draws a line at x = 0. For details, please see **set zeroaxis**.

34.91 Zdata

Set zaxis date to timeseries (dates/times). Please see **set xdata**.

34.92 Zdtics

The **set zdtics** command changes tics on the z axis to days of the week. Please see **set xdtics** for details.

34.93 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.

34.94 Zeroaxis

The x axis may be drawn by **set xzeroaxis** and removed by **set noxzeroaxis**. Similar commands behave similarly for the y, x2, and y2 axes.

Syntax:
```
set {x|x2|y|y2|}zeroaxis { {linestyle | ls <line_style>}
                         | { linetype | lt <line_type>}
                           { linewidth | lw <line_width>}}
set no{x|x2|y|y2|}zeroaxis
show {x|y|}zeroaxis
```

By default, these options are off. The selected zero axis is drawn with a line of type <line_type> and width <line_width> (if supported by the terminal driver currently in use), or a user-defined style <line_style>.

If no linetype is specified, any zero axes selected will be drawn using the axis linetype (linetype 0).

set zeroaxis l is equivalent to **set xzeroaxis l; set yzeroaxis l**. **set nozeroaxis** is equivalent to **set noxzeroaxis; set noyzeroaxis**.

34.95 Zlabel

This command sets the label for the z axis. Please see **set xlabel**.

34.96 Zmtics

The **set zmtics** command changes tics on the z axis to months of the year. Please see **set xmtics** for details.

34.97 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** for details.

34.98 Ztics

The **set ztics** command controls major (labelled) tics on the z axis. Please see **set xtics** for details.

35 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, **endcli** if using AmigaOS, or **exit** if using MS-DOS or OS/2.

A single shell command may be spawned by preceding it with the ! character ($ if using VMS) at the beginning of a command line. Control will return immediately to **gnuplot** after this command is executed. For example, in Unix, AmigaOS, MS-DOS or OS/2,

```
! dir
```

prints a directory listing and then returns to **gnuplot**.

On an Atari, the **!** command first checks whether a shell is already loaded and uses it, if available. This is practical if **gnuplot** is run from **gulam**, for example.

36 Splot

splot is the command for drawing 3-d plots (well, actually projections on a 2-d surface, but you knew that). It can create a plot from functions or a data file in a manner very similar to the **plot** command.

See **plot** for features common to the **plot** command; only differences are discussed in detail here. Note specifically that the **binary** and **matrix** options (discussed under "datafile-modifiers") are not available for **plot**.

Syntax:

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

where either a <function> or the name of a data file enclosed in quotes is supplied. The function can be a mathematical expression, or a triple of mathematical expressions in parametric mode.

By default **splot** draws the xy plane completely below the plotted data. The offset between the lowest ztic and the xy plane can be changed by **set ticslevel**. The orientation of a **splot** projection is controlled by **set view**. See **set view** and **set ticslevel** for more information.

The syntax for setting ranges on the **splot** command is the same as for **plot**. In non-parametric mode, the order in which ranges must be given is **xrange**, **yrange**, and **zrange**. In parametric mode, the order is **urange**, **vrange**, **xrange**, **yrange**, and **zrange**.

The **title** option is the same as in **plot**. The operation of **with** is also the same as in **plot**, except that the plotting styles available to **splot** are limited to **lines**, **points**, **linespoints**, **dots**, and **impulses**; the error-bar capabilities of **plot** are not available for **splot**.

The datafile options have more differences.

36.1 Data-file

As for **plot**, discrete data contained in a file can be displayed by specifying the name of the data file, enclosed in quotes, on the **splot** command line.

Syntax:

```
splot '<file_name>' {binary | matrix}
                    {index <index list>}
                    {every <every list>}
                    {using <using list>}
```

The special filenames "" and "-" are permitted, as in **plot**.

In brief, **binary** and **matrix** indicate that the 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** options **thru** and **smooth** are not available for **splot**, but **cntrparams** and **dgrid3d** provide limited smoothing cabilities.

Data file organization is essentially the same as for **plot**, except that each point is an (x,y,z) triple. If only a single value is provided, it will be used for z, the datablock number will be used for y, and the index of the data point in the datablock will be used for x. If two values are provided, **gnuplot** gives you an error message. Three values are interpreted as an (x,y,z) triple. Additional values are generally used as errors, which can be used by **fit**.

Single blank records separate datablocks in a **splot** datafile; **splot** treats datablocks as the equivalent of function y-isolines. No line will join points separated by a blank record. If all datablocks contain the same number of points, **gnuplot** will draw cross-isolines between datablocks, connecting corresponding points. This is termed "grid data", and is required for drawing a surface, for contouring (**set contour**) and hidden-line removal (**set hidden3d**). See also **splot grid data**

It is no longer necessary to specify **parametric** mode for three-column **splot**s.

36.1.1 Binary

splot can read binary files written with a specific format (and on a system with a compatible binary file representation.)

In previous versions, **gnuplot** dynamically detected binary data files. It is now necessary to specify the keyword **binary** directly after the filename.

Single precision floats are stored in a binary file as follows:

```
<N+1>  <y0>   <y1>   <y2>  ...  <yN>
 <x0> <z0,0> <z0,1> <z0,2> ... <z0,N>
 <x1> <z1,0> <z1,1> <z1,2> ... <z1,N>
  :      :      :      :   ...    :
```

which are converted into triplets:

```
<x0> <y0> <z0,0>
<x0> <y1> <z0,1>
<x0> <y2> <z0,2>
 :    :     :
<x0> <yN> <z0,N>

<x1> <y0> <z1,0>
<x1> <y1> <z1,1>
 :    :     :
```

These triplets are then converted into **gnuplot** iso-curves and then **gnuplot** proceeds in the usual manner to do the rest of the plotting.

A collection of matrix and vector manipulation routines (in C) is provided in **binary.c**. The routine to write binary data is

```
int fwrite_matrix(file,m,nrl,nrl,ncl,nch,row_title,column_title)
```

An example of using these routines is provided in the file **bf_test.c**, which generates binary files for the demo file **demo/binary.dem**.

The **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.

36.1.2 Example datafile

A simple example of plotting a 3-d data file is

```
splot 'datafile.dat'
```

where the file "datafile.dat" might contain:

```
# The valley of the Gnu.
   0 0 10
   0 1 10
   0 2 10

   1 0 10
   1 1 5
   1 2 10

   2 0 10
   2 1 1
   2 2 10

   3 0 10
   3 1 0
   3 2 10
```

Note that "datafile.dat" defines a 4 by 3 grid (4 rows of 3 points each). Rows (datablocks) are separated by blank records.

Note also that the x value is held constant within each dataline. If you instead keep y constant, and plot with hidden-line removal enabled, you will find that the surface is drawn 'inside-out'.

Actually for grid data it is not necessary to keep the x values constant within a datablock, nor is it necessary to keep the same sequence of y values. **gnuplot** requires only that the number of points be the same for each datablock. However since the surface mesh, from which contours are derived, connects sequentially corresponding points, the effect of an irregular grid on a surface plot is unpredictable and should be examined on a case-by-case basis.

36.1.3 Matrix

The **matrix** flag indicates that the ASCII data are stored in matrix format. The z-values are read in a row at a time, i. e.,

```
z11 z12 z13 z14 ...
z21 z22 z23 z24 ...
z31 z32 z33 z34 ...
```

and so forth. The row and column indices are used for the x- and y-values.

36.2 Grid_data

The 3D routines are designed for points in a grid format, with one sample, datapoint, at each mesh intersection; the datapoints may originate from either evaluating a function, see **set isosamples**, or reading a datafile, see **splot datafile**. 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**.

However, **gnuplot** does not require that format. In the case of functions, 'samples' need not be equal to 'isosamples', i.e., not every x-isoline sample need intersect a y-isoline. In the case of data files, if there are an equal number of scattered data points in each datablock, then "isolines" will connect the points in a datablock, and "cross-isolines" will connect the corresponding points in each datablock to generate a "surface". In either case, contour and hidden3d modes may give different plots than if the points were in the intended format. Scattered data can be converted to a {different} grid format with **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 function style 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
```

36.3 Splot_overview

splot can display a surface as a collection of points, or by connecting those points. As with **plot**, the points may be read from a data file or result from evaluation of a function at specified intervals, see **set isosamples**. The surface may be approximated by connecting the points with straight line segments, see **set surface**, 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**) and can then connect those new points to display contour lines, either directly with straight-line segments or smoothed lines (see **set cntrparams**). 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**).

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.

37 Test

test creates a display of line and point styles and other useful things appropriate for the terminal you are using.

Syntax:

```
test
```

38 Update

This command writes the current values of the fit parameters into the given file, formatted as an initial-value file (as described in the **fit**section). This is useful for saving the current values for later use or for restarting a converged or stopped fit.

Syntax:

```
update <filename> {<filename>}
```

If a second filename is supplied, the updated values are written to this file, and the original parameter file is left unmodified.

Otherwise, if the file already exists, **gnuplot** first renames it by appending **.old** and then opens a new file. That is, "**update 'fred'**" behaves the same as "**!rename fred fred.old; update 'fred.old' 'fred'**". [On DOS and other systems that use the twelve-character "filename.ext" naming convention, "ext" will be "**old**" and "filename" will be related (hopefully recognizably) to the initial name. Renaming is not done at all on VMS systems, since they use file-versioning.]

Please see **fit** for more information.

Part III

Graphical User Interfaces

Several graphical user interfaces have been written for **gnuplot** and one for win32 is included in this distribution. In addition, there is a Macintosh interface at

```
ftp://ftp.ee.gatech.edu/pub/mac/gnuplot
```

and several X11 interfaces include three Tcl/Tk located at the usual Tcl/Tk repositories.

Part IV
Bugs

Floating point exceptions (floating point number too large/small, divide by zero, etc.) may occasionally be generated by user defined functions. Some of the demos in particular may cause numbers to exceed the floating point range. Whether the system ignores such exceptions (in which case **gnuplot** labels the corresponding point as undefined) or aborts **gnuplot** depends on the compiler/runtime environment.

The bessel functions do not work for complex arguments.

The gamma function does not work for complex arguments.

As of **gnuplot** version 3.7, all development has been done using ANSI C. With current operating system, compiler, and library releases, the OS specific bugs documented in release 3.5, now relegated to **old_bugs**, may no longer be relevant.

Bugs reported since the current release may be located via the official distribution site:

```
 ftp://ftp.dartmouth.edu/pub/gnuplot
http://www.cs.dartmouth.edu/gnuplot_info.html
```

Please e-mail any bugs to bug-gnuplot@dartmouth.edu.

39 Old_bugs

There is a bug in the stdio library for old Sun operating systems (SunOS Sys4-3.2). The "%g" format for 'printf' sometimes incorrectly prints numbers (e.g., 200000.0 as "2"). Thus, tic mark labels may be incorrect on a Sun4 version of **gnuplot**. A work-around is to rescale the data or use the **set format** command to change the tic mark format to "%7.0f" or some other appropriate format. This appears to have been fixed in SunOS 4.0.

Another bug: On a Sun3 under SunOS 4.0, and on Sun4's under Sys4-3.2 and SunOS 4.0, the 'sscanf' routine incorrectly parses "00 12" with the format "%f %f" and reads 0 and 0 instead of 0 and 12. This affects data input. If the data file contains x coordinates that are zero but are specified like '00', '000', etc, then you will read the wrong y values. Check any data files or upgrade the SunOS. It appears to have been fixed in SunOS 4.1.1.

Suns appear to overflow when calculating exp(-x) for large x, so **gnuplot** gets an undefined result. One work-around is to make a user-defined function like e(x) = x<-500 ? 0 : exp(x). This affects plots of Gaussians (exp(-x*x)) in particular, since x*x grows quite rapidly.

Microsoft C 5.1 has a nasty bug associated with the %g format for 'printf'. When any of the formats "%.2g", "%.1g", "%.0g", "%.g" are used, 'printf' will incorrectly print numbers in the range 1e-4 to 1e-1. Numbers that should be printed in the %e format are incorrectly printed in the %f format, with the wrong number of zeros after the decimal point. To work around this problem, use the %e or %f formats explicitly.

gnuplot, when compiled with Microsoft C, did not work correctly on two VGA displays that were tested. The CGA, EGA and VGA drivers should probably be rewritten to use the Microsoft C graphics library. **gnuplot** compiled with Borland C++ uses the Turbo C graphics drivers and does work correctly with VGA displays.

VAX/VMS 4.7 C compiler release 2.4 also has a poorly implemented %g format for 'printf'. The numbers are printed numerically correct, but may not be in the requested format. The K&R second edition says that for the %g format, %e is used if the exponent is less than -4 or greater than or equal to the precision. The VAX uses %e format if the exponent is less than -1. The VAX appears to take no notice of the precision when deciding whether to use %e or %f for numbers less than 1. To work around this problem, use the %e or %f formats explicitly. From the VAX C 2.4 release notes: e,E,f,F,g,G Result will always contain a decimal point. For g and G, trailing zeros will not be removed from the result.

VAX/VMS 5.2 C compiler release 3.0 has a slightly better implemented %g format than release 2.4, but not much. Trailing decimal points are now removed, but trailing zeros are still not removed from %g numbers in exponential format.

The two preceding problems are actually in the libraries rather than in the compilers. Thus the problems will occur whether **gnuplot** is built using either the DEC compiler or some other one (e.g. the latest gcc).

ULTRIX X11R3 has a bug that causes the X11 driver to display "every other" graph. The bug seems to be fixed in DEC's release of X11R4 so newer releases of ULTRIX don't seem to have the problem. Solutions for older sites include upgrading the X11 libraries (from DEC or direct from MIT) or defining ULTRIX_KLUDGE when compiling the x11.trm file. Note that the kludge is not an ideal fix, however.

The constant HUGE was incorrectly defined in the NeXT OS 2.0 operating system. HUGE should be set to 1e38 in plot.h. This error has been corrected in the 2.1 version of NeXT OS.

Some older models of HP plotters do not have a page eject command 'PG'. The current HPGL driver uses this command in HPGL_reset. This may need to be removed for these plotters. The current PCL5 driver uses HPGL/2 for text as well as graphics. This should be modified to use scalable PCL fonts.

On the Atari version, it is not possible to send output directly to the printer (using **/dev/lp** as output file), since CRs are added to LFs in binary output. As a work-around, write the output to a file and copy it to the printer afterwards using a shell command.

On AIX 4, the literal 'NaNq' in a datafile causes the special internal value 'not-a-number' to be stored, rather than setting an internal 'undefined' flag. A workaround is to use **set missing 'NaNq'**.

There may be an up-to-date list of bugs since the release on the WWW page:

```
http://www.cs.dartmouth.edu/gnuplot_info.html
```

Please report any bugs to bug-gnuplot@dartmouth.edu.