GRAPH++ 3.1
User's Guide

Henning Schulzrinne
Department of Electrical Engineering
Department of Computer and Information Science
University of Massachusetts at Amherst
(hgschulz@cs.umass.edu)

January 9, 1992

1 Legal Matters

The program described in this documentation is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version.

The program is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. See the GNU General Public License for more details. The program should not be relied on when failure or incorrect output could result in injury to a person or loss of property. If you do use the program in such a manner, it is at your own risk. The author disclaims all liability for direct or consequential damages resulting from your use of this program.

2 Introduction

GRAPH^{++} is designed to produce publication-quality line graphs for a number of common output devices. It can be used both interactively and in batch mode. GRAPH^{++} is based on a matrix paradigm, i.e., points and graph descriptions are stored in two-dimensional matrices, similar to MATLAB, MATRIX$_x$, and, to a lesser extent, S.

2.1 Features

GRAPH^{++} offers the following features:

- Available for UNIX[1] (Ultrix, SunOS), VMS[2] and MS-DOS[3].

[1] UNIX is a trademark of AT&T
[2] VMS is a trademark of Digital Equipment Corporation
[3] MS-DOS is a trademark of Microsoft Corporation

- Currently supports PostScript[4], Tektronix[5] 4010/4014, LaTeX EEPIC and UNIXplot(5) output devices. The output can be edited by the IDRAW[6] graphics editor.
- Integrated C preprocessor.
- Data input from wide variety of text files.
- Lines in range between text matching given pattern can be selected.
- Within selected range, lines can be selected and omitted by pattern.
- Number of data points only limited by available virtual memory.
- Matrices can be part of algebraic operations.
- All functions of the C math library can be applied element-by-element to matrices.
- Matrices can be combined into larger matrices.
- Arbitrary submatrices can be extracted.
- Arbitrary submatrices can be assigned to.
- An arbitrary number of graphs can be placed on a page in a rectangular grid.
- Several pages can be generated at once.
- Graphs are sized automatically to fill the maximum amount of space on the page, but can be scaled to any size.
- Bottom and left axes can be different from top and right axes, respectively.
- Unlimited number of axes on each side, each with independent scaling.
- Combinations of linear and logarithmic axes.
- Automatically determines whether linear or logarithmic axis is more appropriate for range of data.
- Labels can be interior to axis, with definable margin at each side.
- Axes labels in fixed point or true exponential notation (shown as 10^5, instead of `10e5`).
- TeX-like label feature for all output devices, allowing for subscript, superscripts, and mixing of typestyles and sizes within a single text string.
- Text labels can be rotated to any angle.
- Confidence intervals, arrows, lassos, circles and boxes can be placed at any data point.

[4] PostScript is a registered trademark of Adobe Systems Incorporated, Palo Alto, California

[5] Tektronix is a trademark of Tektronix Inc., Beaverton, Oregon

[6] IDRAW is available in source form as well as binaries for DECstations and SUN 4's at `interviews.stanford.edu` through anonymous FTP.

- Circles and boxes can be shaded.
- Legends are automatically generated from information provided.
- Legends can be placed anywhere inside or outside the plot area.
- GRAPH^{++} can place legends automatically so as to obscure the least amount of plotted data.
- Produces encapsulated PostScript with proper BoundingBox dimensions, suitable for inclusion in TEX documents.

2.2 Example

An example of a simple graph generated by the program in Fig. 4 is shown in Fig. 3. A more complex graph, with several axes, is shown in Fig. 1, with description in Fig. 2

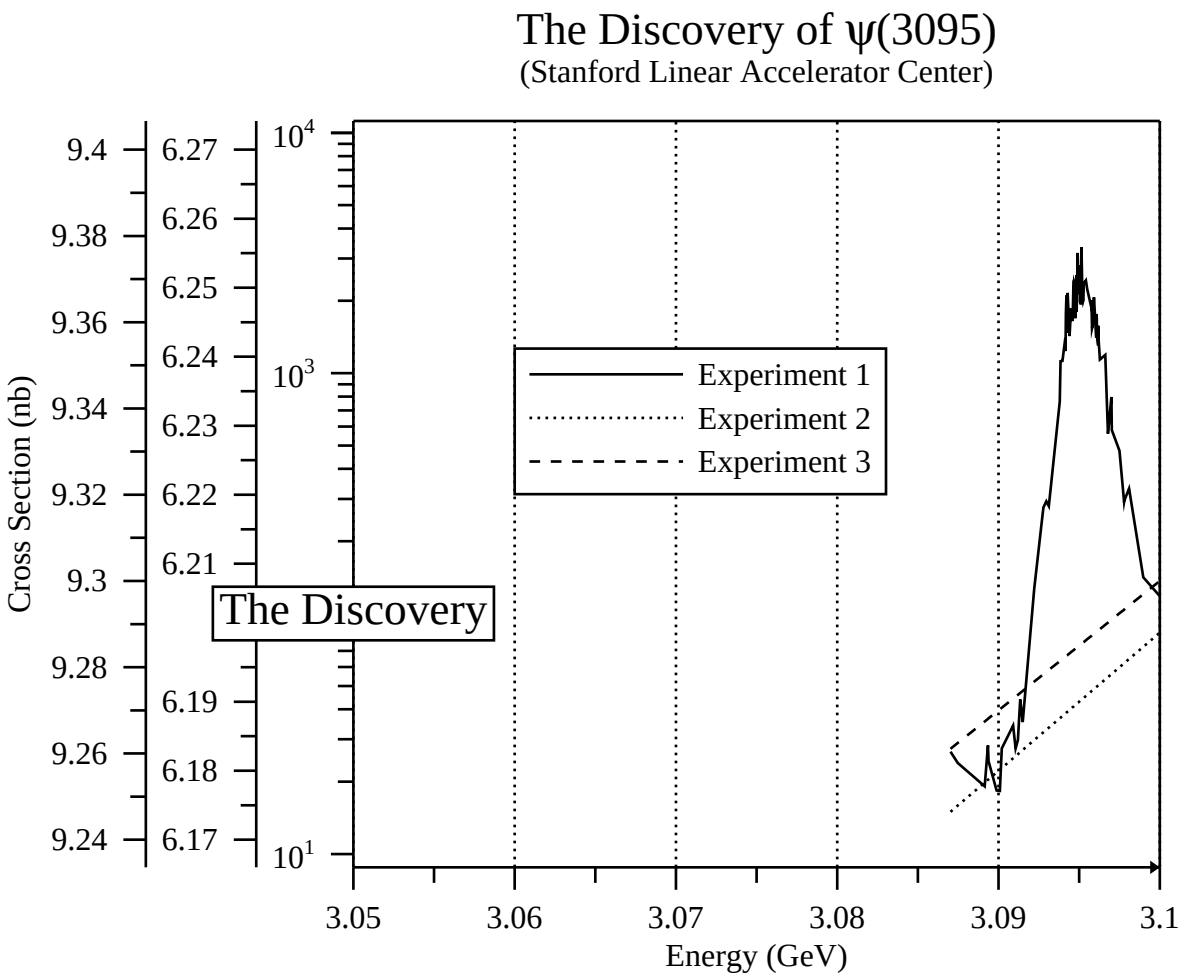

Figure 1: A graph generated by the program in Fig. 2

```
X := read("slac.dat");
page(output="slac",area=[1 1 7 6],layout=[1 1])
graph(legend_at=[0.2,0.5])
plot(X,legend="Experiment 1")
plot(x=X(:,1),y=2*X(:,1),legend="Experiment 2",left,axis=[0,1])
plot(x=X(:,1),y=3*X(:,1),legend="Experiment 3",left,axis=[0,2])
plot([3.05 100],text="\Large The Discovery",box_style="solid",noclip)
label(top,["\Large The Discovery of $\psi$(3095)",
           "(Stanford Linear Accelerator Center)",""])
label(bottom,"Energy (GeV)");
label(left,"Cross Section (nb)")
axis(bottom,arrow,range=[3.05,3.1])
```

Figure 2: The program that generates Fig. 1

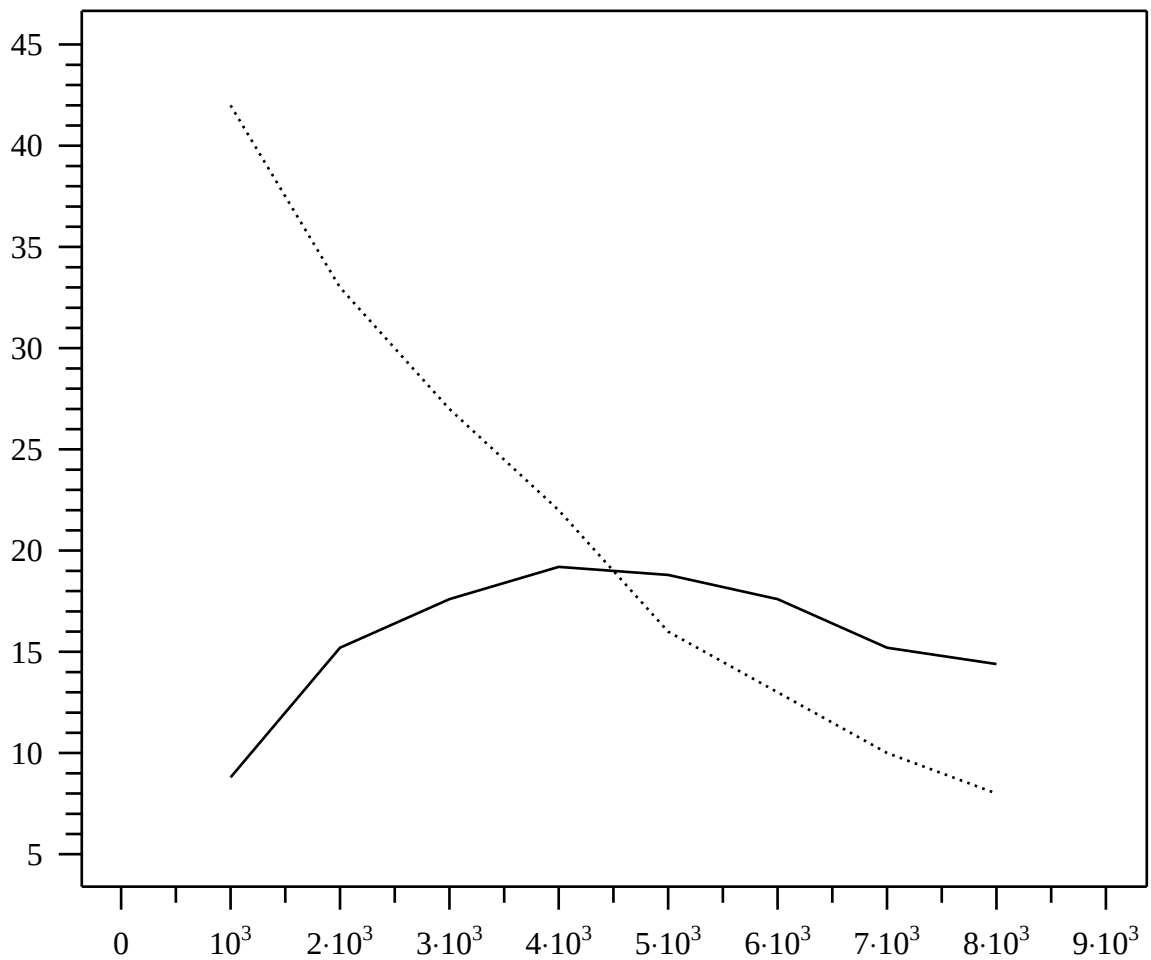

Figure 3: A graph generated by the program in Fig. 4

```
X := [
1000    8.8  42
2000   15.2  33
3000   17.6  27
4000   19.2  22
5000   18.8  16
6000   17.6  13
7000   15.2  10
8000   14.4   8
]
page(prog,area=[1 1 7 6])
plot(X)
```

Figure 4: The program that generates Fig. 3

2.3 Setup and Usage

Before using GRAPH^{++}, the environment variable `AFMDIR` must be properly set, so that it contains the location of the `.afm` font metric files. Ask your system manager or the person who installed GRAPH^{++} where these files are located on your system. Assuming that these files are located in `/usr/lib/tex/fonts/afm`, for a UNIX systems running `csh`, your `.login` should contain the line

```
setenv AFMDIR /usr/lib/tex/fonts/afm
```

For VMS, the corresponding line in your `LOGIN.COM` file would be similar to

```
$ AFMDIR :== TEX:[AFM]
```

The value may already be set for you.

GRAPH^{++} is invoked by typing the following command, where optional parameters are shown enclosed in square brackets. The names of flags and devices can be abbreviated. Flags can be turned on by preceding them with a plus sign, turned off by preceding them with a minus sign. For devices, plus or minus signs are equivalent. Combining several paper sizes will yield unpredictable results, but no error message. In most cases, flags are required only for advanced applications.

`graph++` [*file*] [`+`|`-debug`] [`+`|`-prolog`|`header`] [`+`|`-stroke`] [`+`|`-rotate`] [`+`|`-help`] [`+`|`-overwrite`] [`+11x17` | `+a3` | `+a4` | `+a5` | `+b4` | `+b5` | `+executive` | `+legal` | `+ledger` | `+letter`] [`-tek` | `-ps` | `-unix` | `-eepic` | `-tex`] | `-idraw` [`-D`*name*=*def*]

On VMS and MS-DOS, use `graph` instead of `graph++`. The order of flags is immaterial. On VMS, the file name has to precede all flags. The optional `-debug` flag turns on debugging mode, printing internal data to the standard output.

`prolog` This flag causes GRAPH^{++} to write PostScript, EEPIC and LaTeX output files with a prolog. For PostScript, the prolog contains necessary definitions, but is needed only once for each document. Thus, if the figures are to be included in another document (say, LaTeX), it suffices to explicitly include the header file, `plotps.ps`[7], once at the beginning of the document, saving both storage space for the figures and print time. For `dvips`, for example, include the following at the very beginning of the LaTeX file:

```
\special{header=plotps.ps}
```

For LaTeX and EEPIC, the prolog consists of the necessary definitions so that the output file can be run through LaTeX by itself. This flag has no effect for other output devices and is on by default.

`header` This is a synonym for `prolog`.

`overwrite` Applicable to UNIX and MS-DOS only. The flag determines whether the output graphics file overwrites an existing file of the same name or tries to create a unique name by appending a version number of the form *.n* to the file name. This option is disabled by default. VMS always creates a new version.

`stroke` Use stroked fonts, i.e., where characters are composed of individual lines. Disabling this option invokes the hardware fonts (if available), but may cause errors in text alignment. Stroked text takes up significantly more room than hardware fonts, but not all devices support hardware fonts (see below).

[7] The location of this file may vary; consult the person that installed GRAPH^{++}.

`patterns` Dashed, dotted and other patterned lines can be either generated by the graphics device or in software. The latter yields the same appearance independent of the graphics device and the largest possible number of patterns, but at the cost of significantly larger output files. Enabling this switch has the device generate line patterns. The default depends on the graphics device.

`rotate` Rotate graphics image by 90 degrees.

`letter ...` The paper size can be selected as

`11x17`	11 × 17 inches (B size)
`a3`	297 × 420 mm (A3 size)
`a4`	210 × 297 mm (A4 size)
`a5`	148 × 210 mm (A5 size)
`b4`	257 × 364 mm (B4 size)
`b5`	182 × 257 mm (B5 size)
`executive`	7.5 × 10 inches (executive size)
`ledger`	11 × 17 inches (B size)
`legal`	8.5 × 14 inches (legal size)
`letter`	8.5 × 11 inches (letter size)

Most devices cannot explicitly request paper of different sizes. For all devices, however, the imaging area will be changed, with the assumption that the proper paper will be provided.

`help` Provides a brief summary of the command line options to GRAPH^{++}.

The output device is specified by one of the flags listed below:

`-ps` Encapsulated PostScript output on different paper size and in portrait and landscape orientation. By default, output is on letter-size paper in portrait orientation. In that orientation, the ordinate (y-axis) is parallel to the long edge of the sheet. However, there is one exception. For ledger-size paper, the ordinate is parallel to the short edge of the paper. Note that not all PostScript devices support different paper sizes or trays. This is the default device.

`-idraw` Encapsulated PostScript output on letter-size paper in portrait orientation. The output can also be edited by the IDRAW graphics editor. It runs on X Windows and is available as source through Internet anonymous FTP from node `interviews.stanford.edu`. Binaries for DECstations and SUN 4's can be found there as well. The graphics editor allows easy annotation and addition of rather complicated geometric drawings, including shading and coloring. The output file produced by IDRAW can be directly printed on a PostScript output device or can be included in a TEX file. Output files produced with the `-idraw` option are significantly larger than those written through the `-ps` option.
Since X and PostScript use different fonts, text display suffers from alignment problems. X fonts are only available in a small number of sizes, starting at 8 pts. They are also significantly larger than the corresponding PostScript fonts with the same name. GRAPH^{++} tries to compensate by finding an X font that is closest in size to the PostScript font, but spacing will nevertheless be off. This is particularly noticeable for super- and subscripts. Thus, it is not recommended to construct composite text objects in IDRAW. A translation table to be included

in `.Xdefaults` is provided with the GRAPH^{++} distribution so that text placed within IDRAW follows the same size relation as that generated by GRAPH^{++}. Not all line types are currently supported. Note that, on DECstations, IDRAW will not run properly under the `dxwm` window manager, while the `twm` window manager (and possibly others) work.

`-tek` Tektronix 4010/4014 and other devices emulating them. Currently, the output is scaled and prefixed with the proper escape sequence for a Digital LN03+ laser printer. Other devices may scale differently, so that page frame specifications may not be accurate. Some output devices, for example the LN03+, place the abscissa along the longer side of the sheet, others, for example the DEC ScriptPrinter, along the shorter side. All this implies that Tektronix output may not give a properly scaled representation and that page frames may have to be adjusted. Text written using hardware fonts cannot be rotated and should contain only ASCII characters, without subscripts, etc. Note that the maximum x and y-extents are 10.239 and 7.679, respectively.

`-tex` LaTeX picture-mode output. Due to limitations of LaTeX, rotated text is not supported.

`-eepic` LaTeX picture-mode output using the EEPIC drawing primitives written by Conrad Kwok (University of California at Davis) and available for anonymous FTP at `labrea.stanford.edu`.

`-unix` UNIX `plot(5)` output. The actual device depends on how GRAPH^{++} was linked. Scaling is unpredictable. The same text restrictions as for `tek` apply.

If no device is specified on the command line, the environment variable `GRAPHICS` is used. If the environment variable is undefined, PostScript output is selected.

The default flag settings and suffixes for the graphics output file depend on the output device:

Device	suffix	options	hardware fonts?	sheet
`eepic`	`tex`	`-stroke, -patterns +prolog`	yes	no
`ps`	`ps`	`-stroke, +patterns +prolog`	yes	yes
`idraw`	`ps`	`-stroke, +patterns +prolog`	yes	no
`tek`	`tek`	`+stroke, +patterns`	not recommended	no
`tex`	`tex`	`-stroke, -patterns +prolog`	yes	no
`unix`	`plt`	`+stroke, +patterns`	not recommended	no

Unrecognized flags are passed to the C preprocessor. Due to limitations of the C preprocessor, it is invoked only when GRAPH^{++} is used non-interactively. Particularly useful is the `-D` (UNIX) or `/DEFINE` (VMS) flag that defines preprocessor constants. If strings enclosed in double quotes are to be passed, enclose the flag in single quotes, as shown in the following example.

Example for UNIX:

```
graph++  g.gdl '-DFILE="experiment1"' -noprolog
```

and VMS

```
graph g.gdl -DEFINE "FILE="""experiment1"""""
```

"Space" refers to either a blank or a tab character and is ignored outside of text constants. Numeric constants must begin with a digit, i.e., `.25` is not permitted. Text constants are enclosed in double quotes "; double quotes are inserted into text constants by doubling them. The text "" denotes the empty or null string.

Text bracketed C-style by `/*` and `*/` is treated as a **comment**, as is any text on a line after a double dash `--`. C-style comments can be used only if the preprocessor is active, i.e., if input comes from a file and not the terminal.

Commands are drawn from the input file called *file*; if no file name is specified, input is taken from the standard input. If the standard input is a terminal, GRAPH^{++} prompts with `>` for user input. All standard C preprocessor directives, such as definitions and file inclusions, are supported. The preprocessor is disabled when input comes from a terminal. Keywords and variables are not case sensitive. Normally, the end-of-line (newline, carriage return) terminates a command, but commands can be continued across several lines if the line ends with a backslash (\). If a statement is not complete at the end of a line, the error `Premature end of statement` is signaled.

GRAPH^{++} terminates upon encountering end-of-file or the commands `exit` and `quit`.

3 Matrix Operations

Data in GRAPH^{++} is stored in two-dimensional matrices with real-valued or text entries. Matrix names (variables) can consist of any number of letters, digits and underlines, with lower and upper case being treated as equal. Variable names cannot begin with a digit. Matrices can be either read from a file or entered as constants within the input file. Matrix constants are enclosed in square brackets, with rows separated by semicolons or the end of a line. Columns are separated by commas or space. Negative numbers should always be separated by commas since `[5 -2]` will be interpreted as `[(5-2)]` rather than as a two-column vector. Entries can be constants or expressions, including expressions evaluating to matrices. The result of an operation is printed unless the line ends with a semicolon. It is also assigned to the built-in variable `ans`. See the example shown later.

The tables 6 and 1 through 5 lists the operations that can be performed and their effect on numeric and text matrices. A dash indicates that the operation is not available for that data type. Operations are listed highest precedence first, with operators of equal precedence separated by horizontal lines. Unless otherwise indicated, operations are performed element-by-element; $A, B, \ldots$ indicate expressions evaluating to a matrix. Functions with triple dots in the argument list can take any number of arguments. Relational operator create a matrix of the same size as the matrices to be compared. The resulting matrix has a one at each entry where the relation is true and a zero where it is false. Relational operators on text matrices use lexicographic ordering, with lower-case letters treated as larger than uppercase letters. Functions are equivalent to the C math library function of the same name. All trigonometric functions work in radians. The Bessel and gamma functions are currently not available for the VMS version of GRAPH^{++}.

Matrices to be multiplied, added, subtracted or compared must be compatible in type and size. A matrix and a (scalar) constant can be multiplied, however. In addition, the unary (prefix) positive and negative sign operators are supported.

The **sequence** operator $x_0 : \Delta : x_1$ creates a row vector consisting of the elements $x_0, x_0 + \Delta, x_0 + 2\Delta, \ldots, x_1$. The two-operand form $x_0 : x_1$ is equivalent to $x_0 : 1. : x_1$ if x_0 is less than x_1 and to $x_1 : -1. : x_0$ otherwise. If any of the arguments are matrices, element (1,1) is used.

`all(X)`	row vector; one if corresponding column contain only non-zero values, zero otherwise
`any(X)`	row vector; one if corresponding column contain any non-zero values, zero otherwise
`clock`	vector: $[y, M, d, h, m, s]$, where y stands for the year (e.g., 1991), M for the month, d for the day of month, h for the hour (24 hour clock), m for the minutes and s for the seconds, including fractions of a second on some systems.
`isstr(X)`	zero if matrix X is numeric, one if text
`length(X)`	$\max \text{size}(X)$
`size(X)`	vector containing rows and columns of X
`str2num(T)`	numeric values of text strings
`strlen(T)`	length of text

Table 1: General matrix functions returning numeric values

`acos(X)`	arc cosine
`asin(X)`	arc sine
`atan(X)`	arc tangent
`atan2(Y,X)`	$\arctan(y/x)$ in range $[-\pi, \pi]$
`cos(X)`	cosine
`cosh(X)`	hyperbolic cosine
`hypot(X,y)`	$\sqrt{x^2 + y^2}$
`sin(X)`	sine of x
`sinh(X)`	hyperbolic sine of x
`tan(A)`	tangent of x
`tanh(A)`	hyperbolic tangent of x

Table 2: Trigonometric matrix functions returning numeric values

Subscripts can be applied not only to variables, but to any matrix-valued expression. Subscripts can take two forms. The first, enclosed in parentheses, lists rows and columns, while a matrix applied to an expression selects the elements directly. In the parenthesized form, the colon represents all rows or columns. For example, `A(:,1)` selects the first column of A. Matrices used as subscripts are converted to integer row vectors. If a subscript vector consists of nothing but ones and zeros, it is treated as a logical vector, selecting those elements whose subscript value is one. Matrices of ones and zeros are generated by the relational operators. For example,

```
>a := [4 5 6];
>a(:,[1 2 3] > 1)
5.00 6.00
>a[0 1 0]
5.00
```

3.1 Control Constructs

The GRAPH^{++} programming language provides all standard C [8] and `awk` control constructs, with similar semantics:

[8] except `goto`

`abs(X)`	absolute value; $\|x\|$
`ceil(X)`	smallest integer not less than X
`det(X)`	determinant of X
`erf(X)`	error function: $\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt$
`erfc(X)`	complementary error function, $1 - \mathrm{erf}(x)$
`exp(X)`	exponential function e^x
`finite(X)`	zero if element infinite, one if finite
`fix(X)`	integer part, rounded towards zero
`floor(X)`	largest integer not greater than x
`fmod(X,y)`	remainder of x/y, with sign of x
`gamma(X)`	$\Gamma(x)$
`inf`	the constant "infinity"
`inv(X)`	inverse of matrix; X^{-1}
`j0(X)`	zeroth order Bessel function of first kind
`j1(X)`	first order Bessel function of first kind
`jn(X,n)`	nth order Bessel function of first kind
`ldexp(X,n)`	$x \cdot 2^n$
`log(X)`	natural logarithm, $\ln(x)$
`log10(X)`	decadic logarithm, $\log_{10}(x)$
`na`	numeric constant NA ("Not Available")
`pfit(X,d,b)`	interprets the first column of X as abscissas, the remaining as ordinates and fits a polynomial of degree d (default 1) to the data points in each column and then also evaluates b points in between each abscissa (default 1). The second and third argument can be omitted.
`pi`	the constant π
`pow(X,n)`	x^n
`prod(X)`	row vector containing product of each column of X
`round(X)`	rounded to closest integer
`sqrt(A)`	square root
`sum(X)`	row vector containing sum of each column of X
`y0(X)`	zeroth order Bessel function of second kind
`y1(X)`	first order Bessel function of second kind
`yn(X,n)`	nth order Bessel function of second kind

Notes: Matrix expressions are shown in upper case, e.g., `X`. All functions are applied element-by-element unless otherwise noted. Only the first element is used for arguments shown in lower case.

Table 3: Mathematical matrix functions returning numeric values

The return value of the following functions depends on the type of the argument:

`format(d)`	number of decimals used for printing [2]; returns nothing
`max(A,B,...)`	maximum element
`min(A,B,...)`	minimum element
`sort(X,a,c)`	Sorts according to a given column. The arguments `a` and `c` are optional. `a` designates ascending sort (value of one) or descending sort (value of zero), with a default value of one. `c` determines the column to sort by, with a default value of one.
`system(C)`	pass command C to operating system; returns nothing

Table 4: Matrix functions returning numeric or text values

`chdir(D)`	make `D` current directory; return its name
`date`	date in `dd-mmm-yy` format
`fprintf(N,F,X)`	append `X` formatted as `F` to file `N`
`getenv(E)`	value of environment variable `E`
`int2str(X)`	convert `X` to string, treating it as an integer
`num2str(X)`	convert `X` to string, using roughly four digits of precision
`pause(D)`	wait D seconds
`pause`	wait until key pressed and return character
`prog`	return name of current program, without `.gdl` suffix; (this can be used to generate output files with the same name as the input file)
`sprintf(F,X,...)`	convert up to three numeric arguments to a string according to format `F`

Table 5: Matrix functions returning text values

3.1.1 if (e) *statement*

If expression e is non-zero, *statement* is evaluated.

3.1.2 if (e) *statement1* else *statement2*

If expression e is non-zero, *statement1* is evaluated, otherwise *statement2*.

3.1.3 for (e_0; e_1; e_2) *statement*

The expression e_0 is evaluated before entering the block; expression e_1 is tested before each iteration and should return a boolean value; expression e_2 is evaluated at the bottom of the loop. A missing e_2 makes the implied test equivalent to testing a non-zero constant.

3.1.4 while (e) *statement*

The statement is evaluated repeatedly as long as the value of e remains nonzero. The loop test precedes each iteration.

3.1.5 do *statement* while (e);

The statement is evaluated repeatedly as long as the value of e remains nonzero. The loop test follows each iteration.

3.1.6 break

Can be used in for, do and while loops and unconditionally exits the loop.

3.1.7 continue

Can be used in for, do and while loops and transfer control to the bottom of the loop, that is, just before the loop test.

operator	numeric	text
functions	see Tab. 3 ff.	
`A'`	matrix transpose	
`A(R,C)`	select rows R and columns C from A	
`A(E)`	select elements listed in E from A	
`.*`	element-by-element matrix multiplication	–
`*`	matrix multiplication	–
`./`	element-by-element matrix division	–
`/`	division $A \cdot B^{-1}$	–
`+`	addition	concatenation
`-`	subtraction	–
`>`	greater than	
`<`	less than	
`>=`	greater than or equal	
`<=`	less than or equal	
`=`	equal	
`<>`	not equal	
`:=`	assignment	

Table 6: Operators and Functions

3.2 Examples

1. Creating a two-by-three matrix: `X := [1 2 3;4 5 6]`
2. Replacing all negative elements in the matrix X by zero: `X(X<0) := 0`
3. Selecting odd rows: `X(1:2:3,:)`
4. Replacing the first column in the matrix X (2 rows by 3 columns): `X([1 2],1) := [7;8]`
5. Replacing the first row in the matrix X: `X(1,:) := [7 8 9]`

Errors are shown as in the example below:

```
Undefined variable B.
foo.gdl:1: a := b
                ^
```

In the example, the error occured on line 1. The caret points to the approximate location of the error. If the input comes from the terminal, the file name is listed as `(stdin)`.

For the graphics-related functions described below, **arguments** can be specified by position or keyword (named argument). For example,

```
read(file="in.dat", select=["++","++])
```

is equivalent to

```
read("in.dat",["++","++"])
```

Positional and named arguments can be mixed. Positional arguments are assigned left-to-right to the unassigned named arguments. Any of the arguments can be omitted. Named arguments can be specified in any order.

Predefined constants are a special case of named arguments. They can appear anywhere within the command argument. In the syntax descriptions below, they are shown in all capitals. Example: `axis(left)`. Note that no argument name is specified.

Some arguments will be said to be "converted to vectors". In that case, matrices are treated internally as a vector, with elements appearing in reading order, i.e., with the column index varying fastest.

In descriptions of arguments, default values will be shown in brackets at the end of the description.

4 Text

Text for labels is described in a subset of LaTeX, with similar rules concerning scoping. Within text constants, lower and upper case are distinct. The supported text sizes and fonts are listed in Tab. 7 below, together with the scale factors they apply to the text size. The scale factors are similar to those used by LaTeX. The default is 12 point Roman text. Text enclosed in dollar signs is shown in italics ("math mode"). Unlike TeX, there is no further distinction between regular and math mode, i.e., all symbols, super- and subscripts, etc., are supported in either mode. The greek letters are listed in Tab. 8, while the symbols supported can be found in Tab. 9. Letters from non-English alphabets are listed in Tab. 10. Finally, strings can be accented by the commands given in Tab. 11. The commands marked with a dagger (†) are not supported when generating LaTeX output while others, marked with a double dagger (‡), may require the AMS math symbols. If in doubt, check the LaTeX manual. Note that spacing in LaTeX mode may be off on occasion as GRAPH^{++} can only estimate the width of text.

Name	scale factor
`\tiny`	0.5
`\scriptsize`	0.7
`\footnotesize`	0.8
`\small`	0.9
`\normalsize`	1.0
`\large`	1.2
`\Large`	1.4
`\LARGE`	1.7
`\huge`	2.0
`\Huge`	2.5
`\HUGE`	3.0†

`\rm`	roman
`\bf`	**bold**
`\it`	*italics*
`\bi`	***bold italic***
`\sf`	sans serif
`\sl`	*slanted*
`\tt`	`typewriter`

Table 7: Font sizes and fonts (see text for explanation of †)

Lowercase

α	\alpha	ι	\iota	ϱ	\varrho
β	\beta	κ	\kappa	σ	\sigma
γ	\gamma	λ	\lambda	ς	\varsigma
δ	\delta	μ	\mu	τ	\tau
ϵ	\epsilon	ν	\nu	υ	\upsilon
ε	\varepsilon	ξ	\xi	ϕ	\phi
ζ	\zeta	o	\o	φ	\varphi
η	\eta	π	\pi	χ	\chi
θ	\theta	ϖ	\varpi	ψ	\psi
ϑ	\vartheta	ρ	\rho	ω	\omega

Uppercase

Γ	\Gamma	Ξ	\Phi	Φ	\Phi
Δ	\Delta	Π	\Pi	Ψ	\Psi
Θ	\Theta	Σ	\Sigma	Ω	\Omega
Λ	\Lambda	Υ	\Upsilon		

Table 8: Greek Letters

$\diamondsuit$	\Diamond	°	\degree	†	$\perp$	\perp		
$\Downarrow$	\Downarrow	$\div$	\div		$\pm$	\pm		
$\Im$	\Im	$\downarrow$	\downarrow		$\pounds$	\pounds		
$\Leftarrow$	\Leftarrow	$\emptyset$	\emptyset		$\prod$	\prod		
$\Leftrightarrow$	\Leftrightarrow	$\equiv$	\equiv		$\propto$	\propto		
¶	\P	$\exists$	\exists		$\rangle$	\rangle		
$\Re$	\Re	f	\florin	†	®	\register		‡
$\Rightarrow$	\Rightarrow	$\forall$	\forall		$\rightarrow$	\rightarrow		‡
§	\S	$\geq$	\geq		$\surd$	\root		
$\Uparrow$	\Uparrow	$\heartsuit$	\heartsuit		$\sim$	\sim		
$\aleph$	\aleph	$\in$	\in		$\spadesuit$	\spadesuit		
$\angle$	\angle	∞	\infty		$\subset$	\subset		
$\approx$	\approx	$\int$	\int		$\subseteq$	\subseteq		
*	\ast	$\langle$	\langle		$\sum$	\sum		
$\blacklozenge$	\blacklozenge	‡	...	\ldots	$\supset$	\supset		
•	\bullet		$\leftarrow$	\leftarrow	$\supseteq$	\supseteq		
$\cap$	\cap		$\leftrightarrow$	\leftrightarrow	$\therefore$	\therefore		
$\cdot$	\cdot		$\leq$	\leq	$\times$	\times		
¢	\cent	†	∇	\nabla	™	\trademark		†
$\clubsuit$	\clubsuit		¬	\neg	$\uparrow$	\uparrow		
$\cong$	\cong		$\neq$	\neq	$\vee$	\vee		
©	\copyright		$\notin$	\notin	$\wedge$	\wedge		
$\cup$	\cup		$\not\subset$	\notsubset	$\wp$	\wp		
¤	\currency	†	$\oplus$	\oplus	¥	\yen		‡
†	\dagger		$\otimes$	\otimes				
‡	\ddagger		∂	\partial				

Table 9: Symbols (see text for †, ‡)

œ	\oe	ł	\l
Œ	\OE	Ł	\L
æ	\ae	ß	\ss
Æ	\AE	¡	\!
ø	\o	¿	\?
Ø	\O		

Table 10: Letters from non-English languages

$\acute{a}$	`\acute{a}`	$\check{a}$	`\check{a}`	$\grave{a}$	`\grave{a}`	$\tilde{a}$	`\tilde{a}`
$\bar{a}$	`\bar{a}`	$\ddot{a}$	`\ddot{a}`	$\hat{a}$	`\hat{a}`	$\vec{a}$	`\vec{a}`
$\breve{a}$	`\breve{a}`	$\dot{a}$	`\dot{a}`	$\mathring{a}$	`\ring{a}`		

Table 11: Accents

5 Reading Data from Files: read

```
read(file,select,only,ignore,na,type)
```

read collects numeric data from one or more files into a single matrix. The number of columns is determined by the size of the first valid line. Any non-numeric data is treated as a column separator. The string NA means "Not Available" or "missing".

ARGUMENTS:

file	Each entry is taken to be the name of an input file. If an entry is the null string "", the standard input is used. If the entry begins with a vertical bar (|), the remainder of the entry is interpreted as a shell command and the standard output of that command is read[9].
select	Select range. For each file, only lines between (but not including) lines containing select(1,1) and select(1,2) are considered.
only	Within the select range, only lines containing one of the elements of only are read. If only is not specified, all lines not eliminated by the ignore parameter are read.
ignore	Within the select range, lines containing any of the elements of ignore are ignored. If ignore is not specified, all lines selected by the only parameter are read.
na	Matrix of strings to be used for NaN, the missing-data value [["NA", "NaN"]].
type	Type of data to be read: numeric (real) or character (text) [real].

EXAMPLES:

```
read(file="in.dat",only=["=","+"],select=["SOD","EOD"])
read(file="|spline < sp.dat")
```

6 Plotting Data

Graphs are described hierarchically by a set of commands. An example for a typical sequence of calls is

```
page(...)
  label(...) -- outside labels
  graph(...) -- 1st graph on page
    plot(...) -- 1st plot in graph
    plot(...) -- 2nd plot in graph
    axis(...)
    label(...)
    legend(...)
  graph(...) -- 2nd graph on page
page(...) -- 2nd page
```

[9] For UNIX only

The indentation shows the hierarchy. Since it is a hierarchical description, the order of commands on different levels of the hierarchy does matter, while the order of `plot`, `axis`, `label` and `legend` is immaterial. All of the commands can be repeated or omitted. If commands in the hierarchy are omitted, suitable default values are used. All of these routines return a null matrix.

Note that

```
page(...)
graph(...)
....
```

is quite different from

```
graph(...)
page(...)
...
```

The first creates a page with one graph on it, the second creates two pages, where the first page will be empty, the second page will feature a default graph descriptor.

For axes and labels, the location can be specified either by one of the keywords `bottom`, `left`, `top`, `right` or a numeric argument, with values from one to four corresponding counterclockwise to bottom, left, top and right.

Any line, including within plots, axes, boxes, etc., can be drawn in one of the styles listed below. Here, capitalization is important. Any line can be suppressed by the style `"invisible"` or `""`.

```
"solid" "dotted"  "shortdashed"  "mediumdashed"  "dashed"
"dotdashed"  "dotshortdashed"  "longdotted"  "longdashed"
"Dashed" "invisible"
```

Line widths are expressed in points (1 pt = 1/72" = 0.3527 mm).

6.1 page

`page(output, units, layout, area, extent, textsize, width)`

ARGUMENTS:

`output` Output file. If this is the first page and the argument is missing or equals `""`, the graphics file is sent to the standard output. If the argument is missing for subsequent pages, a new page within the same output file is begun. If the argument begins with an exclamation mark, the remainder of the argument is interpreted as a shell command and the graphics output is piped to the standard input of that command[10]. Use the `prog` function to obtain the name of the current input file.

`units` Flag that determines current measurement units for page area (see below): centimeters (cm), inches (inch) or points (pt) [inch].

[10] For UNIX only.

`layout` Page layout. The first element of the vector determines the number of rows of graphs to be placed on the page, the second the number of graphs per row. Graphs are written left-to-right, top-down. If the number of rows and columns is insufficient to accommodate the number of graphs, then the number of rows is increased as necessary. [By default, graphs are arranged in as close to a square configuration as possible, with more rows than columns if necessary.]

`area` Page area. The four elements of the matrix treated as a vector determine the area on the page used for graphics. The coordinates are measured in inches from the lower left-hand corner, with x values generally along the shorter side of the sheet (This may depend on the device and may not be under control of GRAPH^{++}). Coordinates are specified in the order llx, lly, urx, ury. `llx` and `lly` determine the lower left corner, `urx` and `ury` the upper right corner of the imaging area. Note that legends may fall outside this region.

`extent` Page area as fraction of maximum dimensions. The four elements of the matrix treated as a vector determine the area on the page used for graphics. The coordinates are measured as fractions of the sheet size, in the order noted for `area`. Values should be between 0 and 1, inclusive [1].

`textsize` The default text size, in points, for this page [12].

`width` The default line width, in points, for this page [1].

EXAMPLES:

```
page("g.ps",area=[1,1,8,8])
page("|plot -Tlvp16")
```

6.2 graph

```
graph(legend_at, framestyle, framewidth, legend_line)
```

ARGUMENTS:

`legend_at` Placement of the automatic legend within graph. Positive values give the fraction within the plotting area, negative the fraction within the whole graph. The first value within the vector denotes the horizontal offset, the second the vertical offset. The parameters typically range in absolute values from 0 to 1, with a value of `[0,0]` representing the lower left corner. Values larger than 1 are permissible, allowing legend placement anywhere on the page. Clipping is disabled for legends, so that they may exceed the graph area set within `graph`. If this value is not specified, the legend is placed automatically so that it obscures the minimum amount of plotted information, e.g., lines, text, axes, etc.

`framestyle` Style to be used for the frame around automatic legend. Possible values are listed on p. 21 [`"solid"`].

`framewidth` The line width for the frame around the legend, in points [1].

`legend_line` The length of the line segment within the legend, as a fraction of the horizontal space.

`textsize` The default text size, in points, for this graph [12 or default set by current `page`].

`width` The default line width, in points, for this graph [1] or default set by current `page`].

EXAMPLES:
graph(legend_at=[0.5,0.5],width=2,framestyle="dotted")

6.3 plot

```
plot(x, y, abscissa, ordinate, axis, noclip|clip,
     circle, square, rectangle, cross, confidence, arrow, lasso,
     fill, legend, lines, angle, at, legendsize, width,
     text, textsize, box_style, box_width, box_space)
```

By default, `plot` draws lines connecting the points described by the arguments `x` and `y`. Circles, boxes, etc., can be added at each point, as described below. The ith column of the matrix is used for the ith plot. Columns are reused if necessary. Points where either abscissa or ordinate is `NA` are ignored.

ARGUMENTS:

`x,y` Coordinates of points. If both `x` and `y` are present, column i of matrix Y is plotted against column i of matrix X. If the matrices have different number of columns, columns from the smaller matrix are reused cyclically. If Y is not provided, the first column of X is taken as the abscissa and plotted against the remaining columns.

`abscissa` The position of the abscissa (`bottom` or `top`).

`ordinate` The position of the ordinate (`right` or `left`).

`axis` A vector containing the axis number to be used for abscissa and ordinate. Axes are numbered from the innermost axis to the outside, with the innermost axis having an index of zero. Axis numbers need not be assigned consecutively, but they have to be greater or equal to zero. [0,0]

`noclip` By default, all labels and lines are clipped at the viewport boundaries. By specifying `noclip`, labels and lines can be placed anywhere on the graph. Usually, it will be necessary to freeze the axis range so that the coordinate value given for a label or line does not extend the range [`clip`].

`circle` The diameter of circles, measured along the x-axis.

`square` The side of squares. Note that squares have equal sides only if measured in user coordinates (those shown on the axes), but may appear as rectangles if abscissa and ordinate are scaled differently.

`rectangle` Dimension of rectangles. The first column provides the x dimension, the second column the y dimension.

`cross`	The dimension of crosses, with equal sides.
`confidence`	The halfwidth of confidence intervals. The cross bars are 2% of the x extent of the graph.
`arrow`	The arrow tail is located at the current coordinate. The first column provides the x displacement, the second the y displacement of the arrow head measured from the arrow tail.
`lasso`	A lasso consists of a circle, drawn at the coordinates given by the first and second column and a diameter given by the third column, and an attached line originating at the current coordinate.
`fill`	The gray level for circles, squares and rectangles. The value ranges from zero (black) to one (white) [1].
`legend`	Legend texts. Use "" for plots without legend [""].
`lines`	The style of lines [drawn in sequence from list in Section 6, p. 21; the sequence is maintained across all `plot()` commands within a graph].
`angle`	The text angle for `text`, in degrees [0].
`at`	The relative location of text with respect to the data coordinate. The points are labeled in clock-like fashion. Fig. 5 indicates for each value of `at`, which point on the box surrounding the text will be aligned on the data coordinate.
`legendsize`	The size of the legend, in points [12].
`width`	The width of the plot line, in points [1].
`text`	The text to be shown at each point. This can be used for labels or symbols.
`textsize`	The size of `text`, in points [12].
`box_style`	The style of line used for box surrounding the text (argument `text`), with values drawn from list on p. 21. If text is surrounded by a box, it is also shielded [no box].
`box_width`	The width of the line for box around the text, in mils [13.89].
`box_space`	The space, in mils, between the text and the box surrounding it [20% of the default text size].

6.4 label

```
label(text, side, textsize)
```

`label` is used to place text in the margins of a graph, if used after `graph`, or the page, if used before. Text is written horizontally in the top and bottom margin, with a vertical baseline in the left and right margin. Text is written from the inside out, i.e., the first label command produces text closest to the axis.

ARGUMENTS:

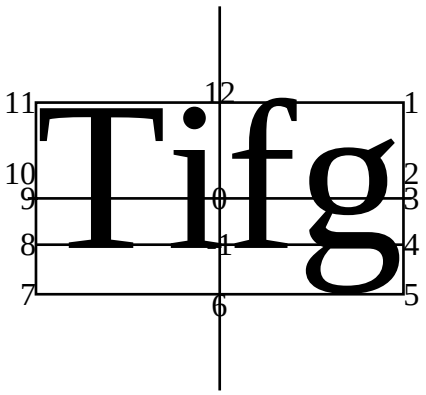

Figure 5: The placement of text

`text`	Matrix containing the label text. Several elements are written from the inside out, i.e., the first element appears closest to the axis.
`side`	The location of the text: `left`, `right`, `top`, `bottom`. Alternatively, the side can be specified numerically, as in `side=1` (see Section 6) [`bottom`].
`textsize`	The size of the text, in points [12].

If `side` or `textsize` have fewer elements than `text`, they are reused cyclically. Matrices are internally converted to vectors.

EXAMPLES:

```
label(side=[1 2],["Bottom label","Left label"],[12,14])
```

6.5 legend

```
legend(at, text, framestyle, framewidth, symbols,
       size, fill, lines, length)
```

`Plot` generates legends automatically from the data supplied in `plot`. If desired, `legend` can be used to add additional legends, each in their own box. Legends consists of rows, each of which contains a line, a symbol placed on the left end of that line and text to the right of the line.

ARGUMENTS:

`at`	Legend placement. See `graph`. By default, legends are placed automatically so that they obscure the least amount of other graphics.
`text`	Matrix containing texts for legends.
`framestyle`	Style of frame around legend, with values drawn from list on p. 21 [`"solid"`].
`framewidth`	Width of frame around legend, in points [1].
`symbols`	Matrix containing the symbols to be placed on the left side of the line [none].

`size` Rows 1 through 4 contain the symbol size, in points [12], the line width, in points [1], the text size, in points [12], and the line length, in horizontal user coordinates. For the line length, only the first column is used, i.e., all lines have the same length.

`fill` Gray shade level for legend box. The value ranges from zero (black) to one (white) [1.0].

`lines` Style of legend lines, with values drawn from list on p. 21 [`invisible`].

`length` The length of the line segment, as a fraction of the horizontal user space.

All matrices are treated as vectors; shorter vectors are reused cyclically.

EXAMPLES:
```
legend([0.5,0.5],text=["first,"second"])
```

6.6 Axis

```
axis(side, format, log|linear,
     axisline, markline, gridline, zeroline, frameline,
     labelsize, marks, submarks, marks_at,
     label_orientation, tick,
     width, range, origin, extend, at, labels,
     space, type, zerospan|nozerospan)
```

ARGUMENTS:

`side` Keyword parameter. The location of the axis: `left`, `right`, `top`, `bottom`. Alternatively, the side can be specified numerically, as in `side=1` (see Section 6) [`bottom`].

`format` Keyword parameter. The format of axis labels. Permissible keywords are

`Exponential`	`%e`
`Fixed`	`%f`
`General`	`%g`
`None`	–

The corresponding `printf` format is shown in the right-hand column. Alternatively, `format=`*number* can be used to specify the format numerically, with zero through two representing fixed, general and exponential format. The general format attempts to print the labels with the minimum number of decimals.

`scaling` Keyword parameter. Scaling of axis, with allowable keywords `log` and `linear` [`linear` unless range of values is too large, then `log`].

`axisline` The style of the axis line. Possible values are listed on p. 21 [`"solid"`].

`markline` The style of the line for marks and submarks. Possible values are listed on p. 21. [`"solid"`].

`gridline` The style of the grid lines. Possible values are listed on p. 21 [`"dotted"` for innermost axes and if that axis is explicitly defined, no grid otherwise].

`zeroline` The style of the zero line ["`solid`" for innermost axis and if that axis is explicitly defined, no zeroline for outer axes and axes that are not explicitly defined].

`frameline` The style of the frame line if the axis is shifted with the `origin` argument; ignored if axis is at the outside of the plotting region ["`solid`"].

`labelsize` The size of the axis labels, in points [12].

`marks` The number of marks along the axis, specified as a two-element vector giving the range of permissible values. The program will attempt to find a number within the range that assures "round" axis labels and extends the axis range the least above the date range [6 ... 12].

`submarks` The number of submarks between each mark, specified as a two-element vector [3 ... 11 for linear axis, 8 for logarithmic axis].

`marks_at` Keyword parameter. The location of marks and submarks; one of `inside`, `outside` or `both` [`outside`].

`label_orientation` The writing direction of axis labels, either `parallel` or `perpendicular` to the axis [`perpendicular` for vertical and `parallel` for horizontal axes].

`tick` The length of the marks, in mils [2% of the width of the drawing region].

`width` Vector containing the width of axis, mark, grid and zero lines, in points [1].

`range` Two-element vector containing the data range [computed from data for this axis].

`origin` The location (constant coordinate) of the axis, in the coordinate system of the adjacent axis (where bottom and left, top and right are considered adjacent, respectively). If the value is NaN, the axis is drawn as a frame [NaN].

`extend` Matrix whose first row provides the percentage extension of the data range before rounding, the second the percentage after rounding. The first column in each row affects the lower end of the range, the second the upper end. Extending before rounding assures that no data point will fall on the data axes, while extending after rounding will create interior labels, that is, labels that are set back from the edge of the axis [all entries 4%]. (Rounding is the process of extending the range so that all labels have round values; round values in our context mean 1, 2 or 5 times integer powers of 10.)

`at` Rather than having GRAPH^{++} compute the location of labels, this vector can be used to explicitly determine their location.

`labels` The text strings used for labels. This can be used for labeling axes with the names of months or other non-numeric data.

`type` Type of axis line, namely `line` or `arrow` [`line`]

`zerospan` For non-negative data, it is often undesirable if the axis is extended to negative values (and, correspondingly, if negative data yields an axis extending into value larger than zero). Thus, by default, the axis can only span zero if the data values

do as well (equivalent to specifying `nozerospan`). The flag `zerospan` allows axis range extension without regard to the sign of the data.

EXAMPLES:
```
axis(left,log,both,marks=[4,6])
```

7 Installation

GRAPH^{++} is written in ANSI C and should compile at least on UNIX and VMSplatforms and possibly MS-DOS. On UNIX systems, Gnu C should be used where available. It has been tested on the following systems:

Hardware	Operating System	Compiler
DECstation 5000	Ultrix 4.0	gcc 1.37.1
VAXstations	Ultrix 3.1, 4.0	gcc 1.37.1
VAX	VMS 5.3, 5.4	DEC C 3.0-031

The installation proceeds as follows:

7.1 UNIX

1. For a source distribution, unpack the file `graph++-V3.00.tar.Z` in a directory `graph++/prog`

   ```
   zcat graph++-V3.00.tar.Z | tar xvf -
   ```

   A directory `doc` will be created that contains LaTeX and PostScript versions of the user documentation.

   For the binary distribution, simply copy the executable `graph++` to a directory that is part of the path and print the PostScript documentation `graph++.ps`.

2. If you don't already have the AFM (Adobe Font Metric) files, unpack the file `afm.tar.Z` with

   ```
   zcat afm.tar.Z | tar xvf -
   ```

   into a convenient directory, for example `/usr/lib/tex/fonts/afm`.

3. Set the environment variable `AFMDIR` to a list of directories where AFM font metric files are to be found. The names of directories are separated by a colon. Example:

   ```
   .:/usr/lib/tex/fonts/afm
   ```

   This declaration will search in the current directory and the `/usr/lib/tex/fonts/afm` directory. The tilde and `~user` notation are expanded to the appropriate login directory as in the C-shell. An empty field expands to the value for `AFMDIR` defined at compile time (typically, `/usr/lib/tex/fonts/afm`).

   Example for UNIX csh:

```
setenv AFMDIR .:/usr/lib/tex/fonts/afm
```

and UNIX Bourne shell (`/bin/sh`:

```
AFMDIR=.:/usr/lib/tex/fonts/afm; export AFMDIR
```

4. For source distribution only: Modify `makefile` to reflect your compiler and binary directory. Also, the definition of `PREPROC` in `graph.c` may need to be changed to correspond to the location of the C (or other macro) preprocessor. Then execute `make install`. On older UNIX systems, compiler warnings about undefined functions `fputs`, `fprintf`, etc. are normal. Object files are unnecessary after the installation is completed; they can be removed by `make clean`. The header file `plotps.ps` should be installed wherever the other header files for dvi-to-PostScript translators (such as `tex.pro` for Tomas Rokicki's `dvips`) are located.

   Code dependencies are marked by the symbols `mips` (currently only used to detect that IEEE floating point arithmetic is available), `unix` and `vms`.

7.2 VMS

1. Modify and execute `VMS_INSTALL.COM`. Add the foreign command `GRAPH` to the user or system wide login file (see example in `VMS_INSTALL.COM`.) Code that depends on features of the VMSoperating system or differs from the UNIX implementation is enclosed in `#ifdef vms` and `#endif` pairs. Currently, a shareable C library cannot be used as it leads to a duplicate definition of `Box` from the `curses` library.

2. Execute the installation file:

   ```
   $ @vms_install
   ```

3. Define the logical name `AFMDIR` so that it points to the directory containing the AFM (Adobe Font Metric) files. A search path consisting of several directories can be specified by separating the directory names with commas. For example, the following definition will cause GRAPH^{++} to search in a system area and the current directory.

   ```
   $ AFMDIR :== SYS$COMMON:[FONTS.AFM],[]
   ```

4. The file `G.COM` can be used to automatically run the C preprocessor on the input files.

8 Reporting Bugs

When reporting bugs, please include a listing of the program file, a script of the terminal messages produced by GRAPH^{++} and a brief description of the problem, if it is not obvious from the terminal script. Also indicate the computer platform used. Shortening data and program files as much as possible will help in isolating the bug.

9 Features waiting to be implemented

The following features are waiting to be implemented, even though they may already be described in this guide:

- multi-line text
- legend symbols in middle of legend line
- tilted boxes
- matrix power
- complex numbers
- eigenvalues and eigenvectors; matrix functions
- on-line help