

Professional Graphics Language

## User Manual

C. Pugmire, St.M. Mundt, V.P. LaBella

<http://glx.sourceforge.net/>

17 March 2005

## Trademark Acknowledgements

The following trademarks are used in this manual.

Windows	Microsoft Corporation.
T <sub>E</sub> X	Donald E. Knuth, A Typesetting System.
L <sup>A</sup> T <sub>E</sub> X	Leslie Lamport, A Document Preparation System.
PKUNZIP	Data decompression software from PKWARE Inc.
PostScript	Page Description Language, Adobe Systems Inc.



# Contents

<b>Preface</b>	<b>v</b>
Abstract . . . . .	v
Typographic Conventions . . . . .	v
Pathways . . . . .	1
<b>1 Tutorial</b>	<b>3</b>
1.1 Installing GLE . . . . .	3
1.2 Running GLE . . . . .	3
1.3 Drawing a Line on a Page . . . . .	4
1.4 Drawing a Simple Graph . . . . .	4
<b>2 Primitives</b>	<b>7</b>
2.1 Graphics Primitives (a summary) . . . . .	7
2.2 Graphics Primitives (in detail) . . . . .	8
2.3 Expressions . . . . .	20
2.4 Functions Inside Expressions . . . . .	21
<b>3 The Graph Module</b>	<b>23</b>
3.1 Graph Commands (a summary) . . . . .	23
3.2 Graph Commands (in detail) . . . . .	24
3.3 Bar Graphs . . . . .	32
3.4 3D Bar Graphs . . . . .	34
3.5 Filling Between Lines . . . . .	34
3.6 Notes on Drawing Graphs . . . . .	35
3.6.1 Importance of Order . . . . .	35
3.6.2 Line Width . . . . .	36
<b>4 The Key Module</b>	<b>37</b>
4.1 Key commands . . . . .	37
4.2 Key syntax . . . . .	38
<b>5 Advanced features</b>	<b>41</b>
5.1 Colour . . . . .	41
5.2 Diagrams, Joining Named Objects . . . . .	42
5.3 L <sup>A</sup> T <sub>E</sub> X Interface . . . . .	43
5.3.1 Example . . . . .	43
5.3.2 Import in a TeX document . . . . .	44
5.3.3 Viewing the Output . . . . .	44

5.3.4	The .gle Directory . . . . .	45
5.4	Filling, Stroking and Clipping Paths . . . . .	45
5.5	Using Variables . . . . .	46
5.6	Programming Loops . . . . .	46
5.7	Extended I/O Functions . . . . .	47
5.8	Device dependend Control . . . . .	47
<b>6</b>	<b>Surface and Contour Plots</b>	<b>49</b>
6.1	Surface Primitives . . . . .	49
6.1.1	Overview . . . . .	49
6.1.2	Surface Commands . . . . .	50
6.2	Letz . . . . .	54
6.3	Fitz . . . . .	55
6.4	Contour . . . . .	56
<b>7</b>	<b>GLE Utilities</b>	<b>59</b>
7.1	Fitls . . . . .	59
7.2	Manip . . . . .	60
7.2.1	Usage . . . . .	61
7.2.2	Manip Primitives (a summary) . . . . .	62
7.2.3	Manip Primitives (in detail) . . . . .	62
<b>A</b>	<b>Tables</b>	<b>69</b>
A.1	Markers . . . . .	69
A.2	Fonts . . . . .	69
A.3	Functions . . . . .	71
A.4	L <sup>A</sup> T <sub>E</sub> X Macros and Symbols . . . . .	73
A.5	Fonttables . . . . .	75
	<b>Wall Reference</b>	<b>77</b>

## Abstract

GLE is a high quality graphics package for scientists, combining a user friendly interface with a full range of facilities for producing publication quality graphs, diagrams, posters and slides.

GLE provides L<sup>A</sup>T<sub>E</sub>X quality fonts together with a flexible graphics module which allows the user to specify any feature of a graph (down to the line width of the subticks, for example)

Complex pictures can be drawn with user defined subroutines and simple looping structures.

Current output file format support: .eps, .ps, .pdf, .svg, .jpg, .png.

GLE runs on Windows, Unix and OS2, giving an identical user interface on all platforms.

## Typographic Conventions

The following conventions will be used in command descriptions:

[option]	Specifies an optional keyword or parameter, the brackets should not be typed.
option1   option2	Pick one of the options listed.
<b>keyword</b>	Keywords are represented in a bold typewriter font.
<i>exp,x,y,x1,y1</i>	Represent numbers or expressions. E.g. 2.2 or 2*5. Parameters to be entered by the user are given in italics.

## Pathways

For those in a hurry:

1. Read chapter 1, GLE Tutorial, (beginners only).
2. Examine the examples at the end of the manual.
3. Browse through Chapter 3, The Graph Module.
4. Read the notes in Chapter 5 on your planned output device.

For those with time:

**Chapter 1, GLE Tutorial:** Covers installation and drawing a simple graph, highly recommended if you have never used GLE before.

**Chapter 2, GLE Primitives:** A detailed description of the commands used for creating slides and annotating graphs.

**Chapter 3, The Graph Module:** A detailed description of the commands for drawing graphs.

**Chapter 4, The Key Module:** This is for producing keys for graphs. Skip this section until you actually want to draw a key.

**Chapter 5, Advanced features of GLE:** Covers programming, filling and clipping. The individual device drivers are also described in this section.

**Chapter 6, GLE Utilities:** Describes FITLS and MANIP.

**Chapter 7, SURFACE:** A detailed description of the commands drawing three-dimensional graphs.

**Examples:** Have a look through these to get an idea of what GLE can do. Most of these are included on the distribution disk and can be used as templates.



# Chapter 1

## Tutorial

### 1.1 Installing GLE

Installation instructions can be found in the ‘readme’ file in the GLE distribution.

Questions, comments and feature requests can be send to the GLE mailing list <https://lists.sourceforge.net/lists/listinfo/glx-general>.

### 1.2 Running GLE

The command to run GLE is:

```
C:\GLE> gle myfile.gle
```

GLE will by default produce an encapsulated PostScript (.eps) file:

```
GLE 4.0.9 [myfile.gle]-C-R-[myfile.eps]
```

Other output formats can be obtained with the -device command line option (which can be abbreviated to -d). For example, to create a jpeg bitmap file, one can use:

```
C:\GLE> gle -d jpg myfile.gle
```

Help about the available command line options can be obtained with:

```
C:\GLE> gle -help
```

and to obtain more information about a particular option, use:

```
C:\GLE> gle -help option
```

### 1.3 Drawing a Line on a Page

Let's start with drawing a line on the page. GLE has to know what size piece of paper you are working with. You can tell it by giving a **size** command:

```
size 18 27
```

This specifies a piece of paper 18cm wide and 27cm high. Now you must define a **current point** by moving to somewhere on the page:

```
amove 2 4
```

The origin (0,0) is at the bottom left hand corner of the page. Now suppose we wish to draw a line from this point across 1 cm and up 2 cm:

```
size 18 27
amove 2 4
rline 1 2
```

That was a **relative** movement as the x and y values were given as distances from the **current point**, alternatively we could have used **absolute** coordinates:

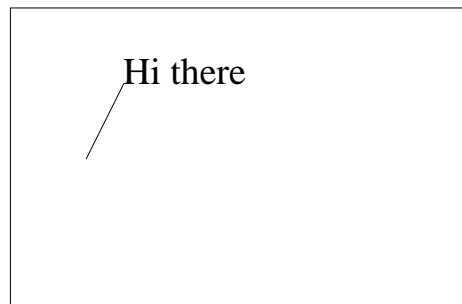
```
size 18 27
amove 2 4
aline 3 6           ! absolute.
```

Now if you want to draw some text on this page at the current point you would use the **text** command:

```
text Hi there
```

So we have now constructed an entire GLE program as follows, with the results illustrated below:

```
size 12 8 box
amove 2 4
rline 1 2
text Hi there
```



### 1.4 Drawing a Simple Graph

This section will describe how to go about drawing a simple graph.

The following data points are contained in a file called TUT.DAT

```

x    y
----- (These top two lines do not appear in the file)
1    2
2    6
3    2
4    5
5    9

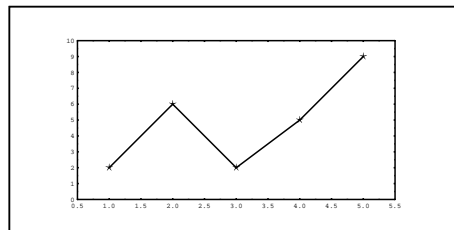
```

The data is in two columns with white space separating each column of numbers. The following commands will draw a simple line graph of the data.

```

size 6 3
begin graph
  size 6 3
  data tut.dat
  yaxis min 0
  d1 line marker star msize .2
end graph

```



The first `size` command defines the size of the piece of paper which may contain several graphs. The second defines the size of this particular graph. The actual graph axes are by default 0.7 of these dimensions. The ratio can be changed with the `yscale` and `hscale` commands.

Without the "`yaxis min 0`" command the yaxis would start at 2.0, because that is the minimum y value in the data. This makes the graph easier to interpret.

Changing the above `d1` command to:

```
d1 line marker circle
```

will mark each data point with a circle instead of a star. See appendix A for a list of other marker names.

A smooth line can be drawn between the data points by changing the `d1` command to:

```
d1 line smooth
```

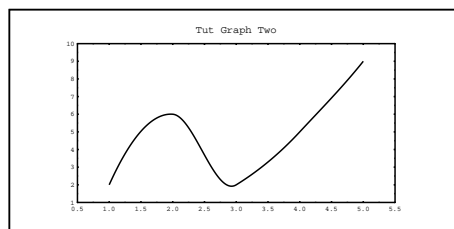
The order of the commands is not important, except that `circle` is a parameter for the qualifier `marker` and therefore must come straight after it. E.g.,

```
d1 [marker circle] [line smooth] [color blue]
```

```

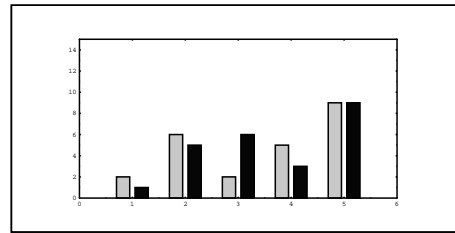
size 6 3
begin graph
  size 6 3
  title "Tut Graph Two"
  data tut.dat
  d1 line smooth
end graph

```



It is simple to change to a bar graph and include last year's measurements:

```
begin graph
  size 6 3
  xaxis min 0 max 6
  yaxis min 0 max 15
  data tut.dat
  data tut2.dat
  bar d1,d2 fill grey10,black
end graph
```



Adding min and max values on the axis commands is highly recommended because by default GLE won't start from  $y=0$  unless the data happens to be very close to zero. It is also impossible to compare graphs unless they all have the same axis ranges.

## Chapter 2

# Primitives

A GLE command is a sequence of keywords and values separated by white space (one or more spaces or tabs). Each command must begin on a new line. Keywords may not be abbreviated, the case is not significant. All coordinates are expressed in centimetres from the bottom left corner of the page.

GLE uses the concept of a **current point** which most commands use. For example, the command `aline 2 3` will draw a line from the **current point** to the coordinates (2,3).

The current graphics state also includes other settings like line width, colour, font, 2d transformation matrix. All of these can be set with various GLE commands.

### 2.1 Graphics Primitives (a summary)

```
! comment
@xxx
aline x y [arrow start] [arrow end] [arrow both]
amove x y
arc radius a1 a2 [arrow end] [arrow start] [arrow both]
arcto x1 y1 x2 y2 rad
begin box [fill pattern] [add gap] [nobox] [name xyz] [round val]
begin clip
begin name
begin origin
begin path [stroke] [fill pattern] [clip]
begin rotate angle
begin scale x y
begin table
begin text [width exp]
begin translate x y
```

```

bezier x1 y1 x2 y2 x3 y3
bitmap filename width height [type type]
bitmap_info filename width height [type type]
box x y [justify jtype] [fill color] [name xxx] [nobox] [round val]
circle radius [fill pattern]
closepath
curve ix iy [x1 y1 x y x y ... xn yn]ex ey
define marker markername subroutine-name
ellipse dx dy [options]
elliptical_arc dx dy theta1 theta2 [options]
for var = exp1 to exp2 [step exp3] command [...] next var
grestore
gsave
if exp then command [...] else command [...] end if
include filename
join object1.just sep object2.just
marker marker-name [scale-factor]
postscript filename.eps width-exp height-exp
print string $ ...
psbttweak
pscomment exp
rbezier x1 y1 x2 y2 x3 y3
return exp
reverse
rline x y [arrow end] [arrow start] [arrow both]
rmove x y
save objectname
set cap butt — round — square
set color col
set dashlen dashlen-exp
set font font-name
set fontlinewidth line-width
set hei character-size
set join mitre — round — bevel
set just left — center — right — tl — etc...
set lstyle line-style
set lwidth line-width
sub sub-name paramter1 paramter2 etc
tex string [name xxx]
text unquoted-text-string
write string $ ...

```

## 2.2 Graphics Primitives (in detail)

! *comment*

Indicates the start of a comment. GLE ignores everything from the exclamation point to the end of the line.

**@xxx**Executes subroutine *xxx*.**aline** *x y* [arrow start] [arrow end] [arrow both]

Draws a line from the current point to the absolute coordinates  $(x,y)$ , which then becomes the new current point. The arrow qualifiers are optional, they draw arrows at the start or end of the line, the size of the arrow is proportional to the current font height.

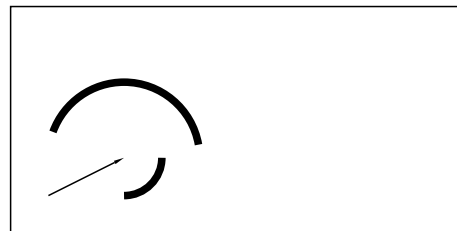
**amove** *x y*Changes the current point to the absolute coordinates  $(x,y)$ .**arc** *radius a1 a2* [arrow end] [arrow start] [arrow both]

Draws an arc of a circle in the anti-clockwise direction, centered at the current point, of radius *radius*, starting at angle *a1* and finishing at angle *a2*. Angles are specified in degrees. Zero degrees is at three o'clock and Ninety degrees is at twelve o'clock.

```
arc 1.2 20 45
```

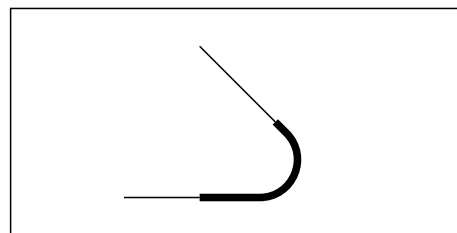
The command **narc** is identical but draws the arc in the clockwise direction. This is important when constructing a path.

```
amove .5 .5
rline 1 .5 arrow end
arc 1 10 160
arc .5 -90 0
```

**arcto** *x1 y1 x2 y2 rad*

Draws a line from the current point to  $(x1,y1)$  then to  $(x2,y2)$  but fits an arc of radius *rad* joining the two vectors instead of a vertex at the point  $(x1,y1)$ .

```
amove 1.5 .5
rline 1 0
set lwidth .1
arcto 2 0 -1 1 .5
set lwidth 0
rline -1 1
```

**begin** *block\_name ... end block\_name*

There are several block structured commands in GLE. Each **begin** must have a matching **end**. Blocks which change the current graphics state (e.g. scale, rotate, clip etc) will restore whatever they change at the end of the block. Indentation is optional but should be used to make the GLE program easier to read.

**begin** **box** [fill *pattern*] [add *gap*] [nobox] [name *xyz*] [round *val*]

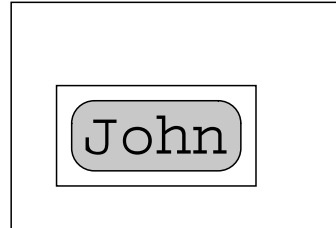
Draws a box around everything between **begin box** and **end box**. The option **add** adds a

margin of **margin** cm to each side of the box to make the box slightly larger than the area defined by the graphics primitives in the **begin box ... end box** group (to leave a gap around text for example). The option **nobox** stops the box outline from being drawn.

The **name** option saves the coordinates of the box for later use with among others the **join** command.

If the **round** option is used, a box with rounded corners will be drawn.

```
amove 1 1
begin box add .2
  begin box fill grey10 add .2 round .3
    text John
  end box
  text John
end box
```



#### **begin clip**

This saves the current clipping region. A clipping region is an arbitrary path made from lines and curves which defines the area on which drawing can occur. This is used to undo the effect of a clipping region defined with the **begin path** command. See the example CLIP.GLE in appendix B at the end of the manual.

#### **begin name**

Saves the coordinates of what is inside the block for later use with among others the **join** command. This command is equivalent to '**begin box name ... nobox**'.

#### **begin origin**

This makes the current point the origin. This is good for subroutines or something which has been drawn using **amove,aline**. Everything between the **begin origin** and **end origin** can be moved as one unit. The current point is also saved and restored.

#### **begin path [stroke] [fill *pattern*] [clip]**

Initialises the drawing of a filled shape. All the lines and curves generated until the next **end path** command will be stored and then used to draw the shape. **stroke** draws the outline of the shape, **fill** paints the inside of the shape in the given colour and **clip** defines the shape as a clipping region for all future drawing. Clipping and filling will only work on PostScript devices.

#### **begin rotate *angle***

The coordinate system is rotated anti-clockwise about the current point by the angle *angle* (in degrees). For example, to draw a line of text running vertically up the page (as a Y axis label, say), type:

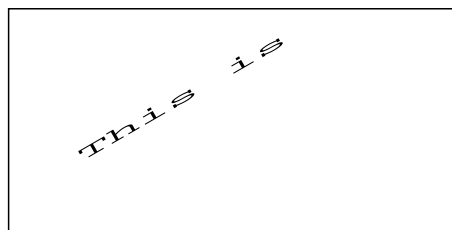
```
amove 1 1
begin rotate 90
  text This is
end rotate
```



**begin scale  $x\ y$** 

Everything between the **begin** and **end** is scaled by the factors  $x$  and  $y$ . E.g., *scale 2 3* would make the picture twice as wide and three times higher.

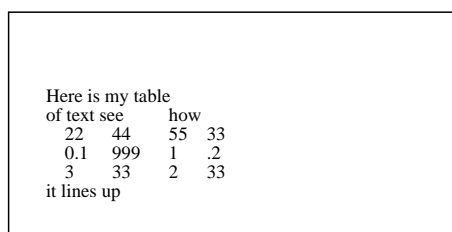
```
amove 1 1
begin scale 3 1
  begin rotate 30
    text This is
  end rotate
end scale
```

**begin table**

This module is an alternative to the **TEXT** module. It reads the spaces and tabs in the source file and aligns the words accordingly. A single space between two words is treated as a real space, not an alignment space.

With a proportionally spaced font columns will line up on the left hand side but not on the right hand side. However with a fixed pitch font, like **tt**, everything will line up.

```
amove .5 .5
set hei .25 just bl
begin table
  Here is my table
  of text see how
    22 44 55 33
    0.1 999 1 .2
    3 33 2 33
  it lines up
end table
```

**begin text [width  $exp$ ]**

This module displays multiple lines/paragraphs of text. The block of text is justified according to the current justify setting. See the **set just** command for a description of justification settings.

If a width is specified the text is wrapped and justified to the given width. If a width is not given, each line of text is drawn as it appears in the file. Remember that GLE treats text in the same way that  $\text{\LaTeX}$  does, so multiple spaces are ignored and some characters have special meaning. E.g.,  $\backslash \wedge \_ \& \{ \}$

To include Greek characters in the middle of text use a backslash followed by the name of the character. E.g.,  $3.3\backslash\Omega\text{ S}$  would produce “3.3ΩS”.

To put a space between the Omega and the S add a backslash space at the end. E.g.,  $3.3\backslash\Omega\backslash\text{ S}$  produces “3.3Ω S”

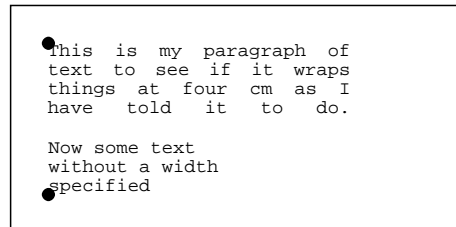
Sometimes the space control characters (e.g.  $\backslash:$ ) are also ignored, this may happen at the beginning of a line of text. In this case use the control sequence **\glass** which will trick GLE into thinking it isn’t at the beginning of a line. E.g.,

```
text \glass \: Indented text
```

```

amove .5 2.5
set hei .25 just tl font tt
begin text width 4
  This is my paragraph of text to see
  if it wraps things at four cm as I have
  told it to do.
end text
amove .5 .5
set justify bl
begin text
  Now some text without
    a width
  specified
end text

```



There are several  $\text{\LaTeX}$  like commands which can be used within text, they are:

$\backslash$ \'	$\backslash$ v \u \= \^	Implemented TeX accents
$\backslash$ . \H \~ \"		
$\wedge\{\}$		Superscript
$\_ \{\}$		Subscript
$\backslash\backslash$		Forced Newline
$\backslash\_$		Underscore character
$\backslash,$		.5em (em = width of the letter 'm')
$\backslash:$		1em space
$\backslash;$		2em space
$\backslash\text{tex}\{\text{expression}\}$		Any LaTeX expression
$\backslash\text{char}\{22\}$		Any character in current font
$\backslash\text{chardef}\{a\}\{\text{hello}\}$		Define a character as a macro
$\backslash\text{def}\backslash v\{\text{hello}\}$		Defines a macro
$\backslash\text{movexy}\{2\}\{3\}$		Moves the current text point
$\backslash\text{glass}$		Makes move/space work on beginning of line
$\backslash\text{rule}\{2\}\{4\}$		Draws a filled in box, 2cm by 4cm
$\backslash\text{setfont}\{\text{rmb}\}$		Sets the current text font
$\backslash\text{sethei}\{.3\}$		Sets the font height (in cm)
$\backslash\text{setstretch}\{2\}$		Scales the quantity of glue between words
$\backslash\text{lineskip}\{.1\}$		Sets the default distance between lines of text
$\backslash\text{linegap}\{-1\}$		Sets the minimum required gap between lines

**begin** *translate*  $x$   $y$

Everything between the **begin** and **end** is moved  $x$  units to the right and  $y$  units up.

**bezier**  $x_1$   $y_1$   $x_2$   $y_2$   $x_3$   $y_3$

Draws a Bézier cubic section from the current point to the point  $(x_3, y_3)$  with Bézier cubic control points at the coordinates  $(x_1, y_1)$  and  $(x_2, y_2)$ . For a full explanation of Bézier curves see the PostScript Language Reference Manual.

**bitmap** *filename* *width* *height* [*type type*]

Imports the bitmap *filename*. The bitmap is scaled to  $\text{width} \times \text{height}$ . If one of these is zero, it is computed based on the other one and the aspect ratio of the bitmap. GLE supports TIFF, JPEG, PNG and GIF bitmaps (depending on the compilation options).

Bitmaps are compressed automatically by GLE using either the LZW or the JPEG compression scheme.



The second parameter can be supplied using the *MDATA* command when drawing a graph, this gives the marker subroutine a value from another dataset to use to draw the marker. For example the marker could vary in size, or angle, with every one plotted.

```
d3 MARKER myname MDATA d4
```

```
define markername fontname scale dx dy
```

This command defines a new marker, from any font, it is automatically centered but can be adjusted using *dx*,*dy*. e.g.

```
defmarker hand pszd 43 1 0 0
```

```
ellipse dx dy [options]
```

This command draws an ellipse with the diameters *dx* and *dy* in the *x* and *y* directions, respectively. The *options* are the same as the *circle* command.

```
elliptical_arc dx dy theta1 theta2 [options]
```

This command is similar to the *arc* command except that it draws an elliptical arc in the clockwise direction with the diameters *dx* and *dy* in the *x* and *y* directions, respectively. *theta1* and *theta2* are the start and stop angle, respectively. The *options* are the same as for the *arc* command.

The command *elliptical\_narc* is identical but draws the arc in the clockwise direction. This is important when constructing a path.

```
for var = exp1 to exp2 [step exp3] command [...] next var
```

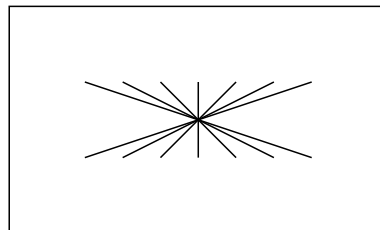
The *for ... next* structure lets you repeat a block of statements a number of times.

GLE sets *var* equal to *exp1* and then repeats the following steps.

- If *var* is greater than *exp2* then GLE commands are skipped until the line after the *next* statement.
- The value *exp3* is added to *var*.
- The statements between the *for* and *next* statement are executed.

If *exp1* is greater than *exp2* then the loop is not executed.

```
for xx = 1 to 4 step .5
  amove xx 1
  aline 5-xx 2
next xx
```



```
grestore
```

Restores the most recently saved graphics state. This is the simplest way to restore complicated transformations such as rotations and translations. It must be paired with a previous *gsave* command.

**gsave**

Saves the current graphics transformation matrix and the current point and the current colour, font etc.

**if *expression* then *command* [...] else *command* [...] end if**

If *expression* evaluates to true, then execution continues with the statements up to the corresponding **else**, otherwise the statements following the **else** and up to the corresponding **end if** are executed.

```

amove 3 3
if xpos()=3 then
    text We are at x=3
else
    text We are elsewhere
end if

```

Note: **end if** is not spelt **endif**.

**include *filename***

The commands in *filename* are read just as if they were in the current GLE file. With a large include file GLE may run out of memory. If this happens, use the **bigfile** command instead of **include**. Note: there is also a **bigfile** option in the graphing module.

**join *object1.just sep object2.just***

Draws a line between two named objects. An object is simply a point or a box which was given a name when it was drawn.

The justify qualifiers are the standard GLE justification abbreviations (e.g. TL=top left, see **set justify** for details)

If *sep* is written as -, a line is drawn between the named objects e.g.

```

join fred.tr - mary.tl

```

Arrow heads can be included at both ends of the line by writing *sep* as <->. Single arrow heads are produced by <- and ->. Note that *sep* must be separated from *object1.just* and *object2.just* by white space.

If the justification qualifiers are omitted, a line will be drawn between the centres of the two objects (clipped at the edges of the rectangles which define the objects).

See Chapter 5 for an example of joining objects.

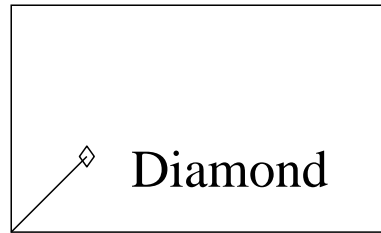
**marker *marker-name* [*scale-factor*]**

Draws marker *marker-name* at the current point. The size of the marker is proportional to the current font size, scaled by the value of *scale-factor* if present. Markers are referred to by name, eg. **square**, **diamond**, **triangle** and **fcircle**. Markers beginning with the letter **f** are usually filled variants. Markers beginning with **w** are filled with white so lines are not visible through the marker. For a complete list of markers refer to Appendix A.1.

```

line 1 1
marker diamond 1
rmove .6 -.4
set font rm hei .7
text Diamond

```



**postscript** *filename.eps width-exp height-exp*

Includes an encapsulated postscript file into a GLE picture, the postscript picture will be scaled up or down to fit the width given. On the screen you will just see a rectangle.

Only the *width-exp* is used to scale the picture so that the aspect ratio is maintained. The height is only used to display a rectangle of the right size on the screen.

**print** *string\$* ...

This command prints its argument to the console (terminal).

**psbbtweak**

Changes the default behavior of the bounding box. The default behavior is to have the lower corner at (-1,-1), which for some interpreters (i.e., Photoshop) will leave a black line around the bottom and left borders. If this command is specified then the origin of the bounding box will be set to (0,0).

This command must appear before the first **size** command in the GLE file.

**pscomment** *exp*

Allows inclusion of *exp* as a comment in the preamble of the postscript file. Multiple **pscomment** commands are allowed.

This command must appear before the first **size** command in the GLE file.

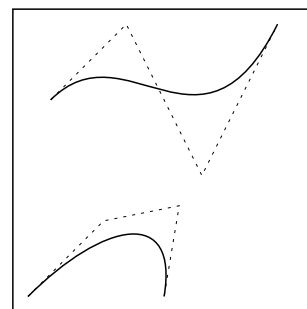
**rbezier** *x1 y1 x2 y2 x3 y3*

This command is identical to the **BEZIER** command except that the points are all relative to the current point.

```

amove .5 2.8
rbezier 1 1 2 -1 3 1
amove .2 .2
rbezier 1 1 2 1.2 1.8 0

```



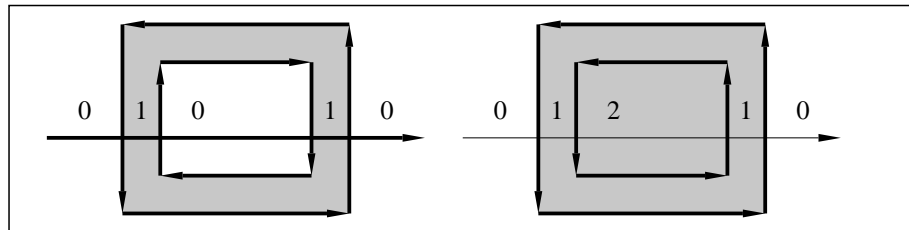
**return** *exp*

The **return** command is used inside subroutines to return a value.

**reverse**

Reverses the direction of the current path. This is used when filling multiple paths in order that the Non-Zero Winding Rule will know which part of the path is 'inside'.

With the Non-Zero Winding Rule an imaginary line is drawn through the object. Every time a line of the object crosses it from left to right, one is added to the counter; every time a line of the object crosses it from right to left, one is subtracted from the counter. Everywhere the counter is non-zero is considered to be the ‘inside’ of the drawing and is filled.



**rline** *x y* [arrow end] [arrow start] [arrow both]

Draws a line from the current point to the relative coordinates  $(x,y)$ , which then become the new current point. If the current point is (5,5) then **rline** 3 -2 is equivalent to **aline** 8 3. The optional qualifiers on the end of the command will draw arrows at one or both ends of the line, the size of the arrow head is proportional to the current font size.

**rmove** *x y*

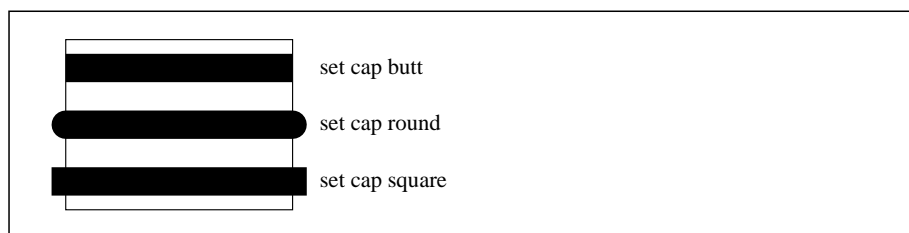
Changes the current point to the relative coordinate  $(x,y)$ . If the current point is (5,5) then **rmove** 3 -2 is equivalent to **amove** 8 3.

**save** *objectname*

This command saves a point for later use with the join command.

**set cap** butt — round — square

Defines what happens at the end of a wide line.



**set color** *col*

Sets the current colour for all future drawing operations. There are several pre-defined colours which can be specified by name.

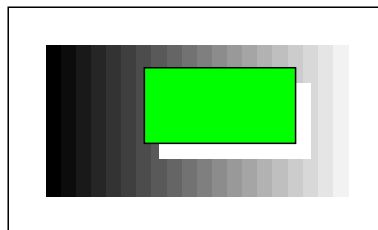
black, white, red, green, blue, cyan, magenta, yellow, grey10, grey20 ... grey90, shade1 ... shade5, grid1 ... grid5

It is also possible to specify a grey scale as an expression with 0.0 = black and 1.0 = white.

```

mm$ = "green"
amove .5 .5
for c = 0 to 1 step .05
  box .2 2 fill (c) nobox
  rmove .2 0
next c
amove 2 1
box 2 1 fill white nobox
rmove -.2 .2
box 2 1 fill mm$

```



### set dashlen *dashlen-exp*

Sets the length of the smallest dash used for the line styles. This command **MUST** come before the `set lstyle` command. This may be needed when scaling a drawing by a large factor.

### set font *font-name*

Sets the current font to *font-name*. Valid *font-names* are listed in Appendix A.2.

There are three types of font: PostScript,  $\text{\LaTeX}$  and Plotter. They will all work on any device, however  $\text{\LaTeX}$  fonts are drawn in outline on a plotter, and so may not look very nice. PostScript fonts will be emulated by  $\text{\LaTeX}$  fonts on non-PostScript printers.

### set fontlwidth *line-width*

This sets the width of lines to be used to draw the stroked (Plotter fonts) on a PostScript printer. This has a great effect on their appearance.

```

set font pltr
amove .2 .2
text Tester
set fontlwidth .1
set cap round
rmove 0 1.5
text Tester

```

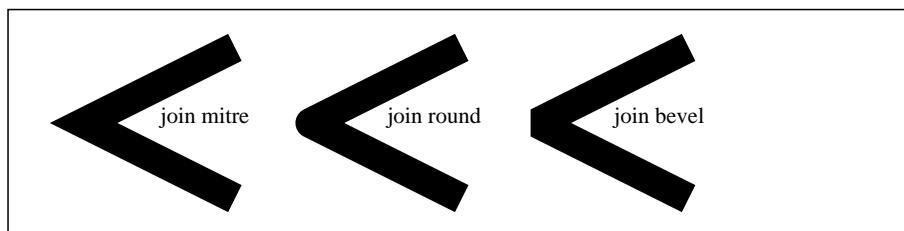


### set hei *character-size*

Sets the height of text. For historical reasons, concerning lead type and printing conventions, a height of 10cm actually results in capital letters about 6.5cm tall.

### set join mitre — round — bevel

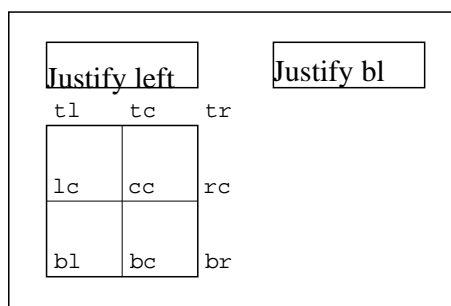
Defines how two wide lines will be joined together. With **mitre**, the outside edges of the join are extended to a point and then chopped off at a certain distance from the intersection of the two lines. With **round**, a curve is drawn between the outside edges.



`set just left` — `center` — `right` — `tl` — `etc...`

Sets the justification which will be used for text commands.

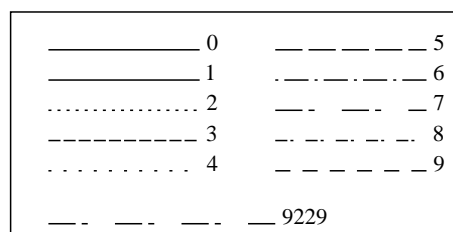
```
amove .5 3
set just left
box 1.5 .6
text Justify left
rmov 2 0
set just bl
box 1.5 .6
text Justify bl
```



`set lstyle` *line-style*

Sets the current line style to line style number *line-style*. There are 9 predefined line styles (1–9). When a line style is given with more than one digit the first digit is read as a run length in black, the second a run length in white, the third a run length in black, etc.

```
set hei .3
set just left
amove .5 2.5
for z = 0 to 4
  set lstyle z
  rline 2 0
  rmov .1 0
  write z
  rmov -2.1 -.4
next z
```



`set lwidth` *line-width*

Sets the width of lines to *line-width* cm. A value of zero will result in the device default of about 0.02 cm, so a `lwidth` of .0001 gives a thinner line than an `lwidth` of 0.

`sub` *sub-name* *parameter1* *parameter2* *etc*

Defines a subroutine. The end of the subroutine is denoted with `end sub`. Subroutines must be defined before they are used.

Subroutines can be called inside any GLE expression, and can also return values. The parameters of a subroutine become local variables. Subroutines are reentrant.

```
sub tree x y a$
  amove x y
  rline 0 1
  write a$
  return x/y
end sub
@tree 2 4 "mytree"          (Normal call to subroutine)
slope = tree(2,4,"mytree")  (Using subroutine in an expression)
```

`tex` *string* [*name xxx*]

Draw a  $\text{\LaTeX}$  expression at the current point using the current value of 'justify'. See Section 5.3 for more information. Using the `name` option, the  $\text{\LaTeX}$  expression can be named, just like a box.

`text` *unquoted-text-string*

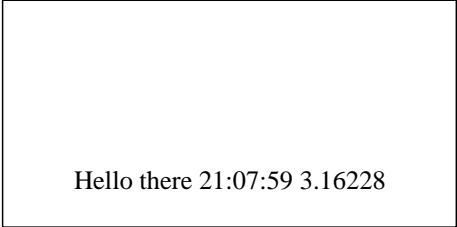
This is the simplest command for drawing text. The current point is unmodified after the text is drawn so following one text command with another will result in the second line of text being drawn on top of the first. To generate multiple lines of text, use the `begin text ...end text` construct.

```
text "Hi, how's tricks", said Jack!
```

`write` *string\$* ...

This command is similar to `text` except that it expects a quoted string, string variable, or string expression as a parameter. If `write` has more than one parameter, it will concatenate the values of all the parameters.

```
set hei .35 font rm
set just center
amove 3 .5
a$ = "Hello there "
xx = sqrt(10)
t$ = time$()
c$ = a$+t$
write a$+t$ xx
```



Hello there 21:07:59 3.16228

The built in functions `sqrt()` and `time$()` are described in Appendix A.3.

## 2.3 Expressions

Wherever GLE is expecting a number it can be replaced with an expression. For example

```
rline 3 2
```

and

```
rline 9/3 sqrt(4)
```

will produce the same result.

An expression in GLE is delimited by white space, so it may not contain any spaces - '`rline 3*3 2`' is valid but '`rline 3 * 3 2`' will not work.

Or '`let d2 = 3+sin(d1)`' will work and '`let d2= 3 + sin(d1)`' won't.

Expressions may contain numbers, arithmetic operators (+, -, \*, /, ^ (to the power of)), relational operators (>, <, =, <=, >=, <>) boolean operators (AND, OR), variables and built-in functions.

When GLE is expecting a colour or marker name (like 'green' or 'circle') it can be given a string variable, or an expression enclosed in braces).

## 2.4 Functions Inside Expressions

`format$(exp,format)`

Returns a string representation of *exp* formatted as specified in *format*.

Basic formats:

- `dec`, `hex` [`upper—lower`], `bin`: format as decimal, hexadecimal (upper-case or lower-case), or binary.
- `fix places`: format with *places* decimal places.
- `sci sig [e,E,10] [expdigits num] [expsign]`: format in scientific notation with *sig* significant digits. Use 'e', 'E', or '10' as notation for the exponent. With the option `expdigits` the number of digits in the exponent can be set and `expsign` forces a sign in the exponent.
- `round sig`: format a number with *sig* significant digits.

Options for all formats:

- `nozeroes`: remove unnecessary zeroes at the end of the number.
- `sign`: also include a sign for positive numbers.
- `pad nb [left] [right]`: pad the result with spaces from the left or right.
- `prefix nb`: prefix the number with zeroes so that *nb* digits are obtained.
- `min val`: use format for numbers  $\geq val$ .
- `max val`: use format for numbers  $\leq val$ .

Examples:

- `format$(3.1415, "fix 2") = 3.14`
- `format$(3756, "round 2") = 3800`
- `format$(3756, "sci 2 10 expdigits 2") = 3.8 · 1003`

Several formats can be combined into one string: "sci 2 10 min 1e2 fix 0" uses scientific notations for numbers above 10<sup>2</sup> and decimal notation for smaller numbers.

`pagewidth()`, `pageheight()`

These functions return the width and height of the output. These are the values set with the 'size' command.

`pointx()`, `pointy()`

These functions return the x and y values of a named point.

```
begin box add 0.1 name mybox
  write "Hello"
end box
amove pointx(mybox.bc) pointy(mybox.bc)
rline 0 -2 arrow end
```

`twidth()`, `theight()`, `tdepth()`

These functions return the width, depth and height of a string, if it was printed in the current font and size.

`width()`, `height()`

These functions return the width and height of a named object.

`xend()`, `yend()`

These functions return the end point of the last thing drawn. This is of particular interest when drawing text.

```
text abc
set color blue
text def
```

This would draw the `def` on top of the `abc`. To draw the `def` immediately following the `abc` simply do the following (Note that absolute move is used, not relative move):

```
set hei 1 font rm
set just left
amove 1 1
text abc
set color grey20
amove xend() yend()
text def
```



`xg()`, `yg()`

With these functions it is possible to move to a position on a graph using the graph's axis units. To draw a filled box on a graph, at position `x=948`, `y=.004` measured on the graph axis:

```
begin graph
  xaxis min 100 max 2000
  yaxis min -.01 max .01
  ...
end graph
amove xg(948) yg(.004)
box 2 2 fill grey10
```

`xpos()`, `ypos()`

Returns the current x and y points.

See Appendix A.3 for an overview of all functions provided by GLE.

## Chapter 3

# The Graph Module

A graph should start with `begin graph` and end with `end graph`. The data to be plotted are organised into datasets. A dataset consists of a series of (X,Y) coordinates, and has a name based on the letter “d” and a number between 1 and 99, eg. d1

The name `dn` can be used to define a default for all datasets. Many graph commands described below start with `dn`. This would normally be replaced by a specific dataset number e.g.,

```
d3 marker diamond
```

For each `xaxis` command there is a corresponding `yaxis`, `y2axis` and `x2axis` command for setting the top left and right hand axes. These commands are not explicitly mentioned in the following descriptions.

### 3.1 Graph Commands (a summary)

```
data filename [d1 d2 d3 ...] [d1=c1,c3] [ignore n]
dn bigfile "all.dat,xc,yc" [marker mname] [line]
dn bigfile xxx$ [autoscale]
dn err d5 errwidth width-exp errup nn% errdown d4
dn herr d5 herrwidth width-exp herrleft nn% errright d4
dn key "Dataset title"
dn line [impulses] [steps] [fsteps] [hist] [svg_smooth]
dn lstyle line-style lwidth line-width color col
dn marker marker-name [msize marker-size] [mdata dn]
dn nomiss
dn smooth — smoothm
dn xmin x-low xmax x-high ymin y-low ymax y-high
fullsize
hscale exp
key pos tl nobox hei exp offset xexp yexp
```

```

let ds = exp [from low to high step exp]
let dn = [routine] dm [options]
nobox
size x y
title "title" [hei ch-hei] [color col] [font font] [dist cm]
vscale exp
x2labels on
xaxis — yaxis — x2axis — y2axis
xaxis base exp-cm
xaxis color col font font-name hei exp-cm lwidth exp-cm
xaxis dsubticks sub-distance
xaxis format format-string
xaxis grid
xaxis log
xaxis min low max high
xaxis nofirst nolast
xaxis nticks number dticks distance dsubticks distance
xaxis off
xaxis shift cm-exp
xlabel font font-name hei char-hei color col
xnames "name" "name" ...
xplaces pos1 pos2 pos3 ...
xside color col lwidth line-width off
xsubticks lstyle num lwidth exp length exp off
xticks lstyle num lwidth exp length exp off
xtitle "title" [hei ch-hei] [color col] [font font] [dist cm]
y2title "text-string" [rotate]
yaxis negate
bar dx,... dist spacing
bar dx,... from dy,...
bar dn,... width xunits,... fill col,... color col,...
fill x1,d3 color green xmin val xmax val
fill d4,x2 color blue ymin val ymax val
fill d3,d4 color green xmin val xmax val
fill d4 color green xmin val xmax val

```

## 3.2 Graph Commands (in detail)

**data** *filename* [*d1 d2 d3 ...*] [*d1=c1,c3*] [ignore *n*]

Specifies the name of a file to read data from. By default, the data will be read into the next free datasets unless the optional specific dataset names are specified.

A dataset consists of a series of (X,Y) coordinates, and has a name based on the letter d and a number between 1 and 99, e.g. d1 or d4. Up to 99 datasets may be defined.

From a file with 3 columns the command ‘data xx.dat’ would read the first and second columns as the x and y values for dataset 1 (d1) and the first and third columns as the x and y values for dataset 2 (d2). The next data command would use dataset 3 (d3).

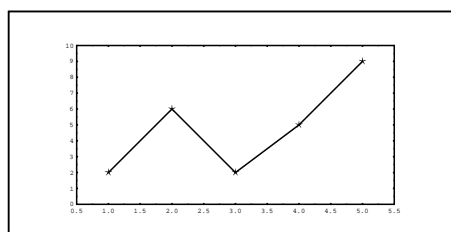
A data file for two datasets looks like this (\*=missing value):

```
1  2.7  3
2  5    *
3  7.8  7
4  9    4
```

The first coordinate of dataset **d1** would then be (1,2.7) and the first coordinate of dataset **d2** would be (1,3).

The option **d3=c2,c3** allows particular columns of data to be read into a dataset, **d3** would read x values from column 2 and y values from column 3.

```
size 6 3
begin graph
  size 6 3
  data tut.dat
  yaxis min 0
  d1 line marker star msize .2
end graph
```



The option **ignore n** makes GLE ignore the first *n* lines of the data file. This is useful if the first *n* lines contain attribute names/types.

**dn bigfile "all.dat,xc,yc" [marker mname] [line]**

The **bigfile** option allows a dataset to be read as it is drawn, (rather than being complete read into memory before it is drawn) this means that very large datasets can be drawn on a PC without running out of memory. The axis minimum and maximum must be specified (using the command **xaxis min exp max exp**).

By default the first two columns of the data file will be read in, but other columns may be specified. E.g., **all.dat,3,2** would read x values from column 3 and y values from column 2. Or, to read the 4th dataset, specify the file as **all.dat,1,5**

If the x column is specified as '0' then GLE will generate the x data points. E.g., 1,2,3,4,5...

**Bigfile** also accepts variables in place of the file name, e.g.

```
xxx$ = "test.dat,2,3"
d1 bigfile xxx$
```

The **AUTOSCALE** option pre-reads the file to scale the axis, which is slow but sometimes required, e.g.:

```
d1 bfile a.dat line autoscale
```

Many (but not all) of the normal **dn** commands can be used with the **bigfile** command. E.g., **marker**, **lstyle**, **xmin**, **xmax**, **ymin**, **ymax**, **color** and **lwidth**. You cannot use commands like **let** or **bar** with the **bigfile** command.

**dn err d5 errwidth width-exp dn errup nn% errdown d4**

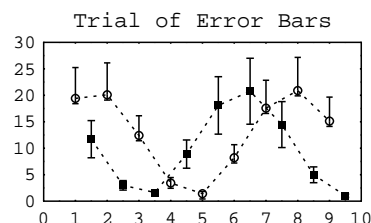
For drawing error bars on a graph. The error bars can be specified as an absolute value, as a percentage of the y value, or as a dataset. The up and down error bars can be specified separately e.g.,

```

d3 err .1
d3 err 10%
d3 errup 10% errdown d2
d3 err d1 errwidth .2

begin graph
  size 6 3
  title "Trial of Error Bars"
  yaxis min 0 max 30
  dn lstyle 2 msize 1.5
  d1 marker circle errup 30%
    errdown 1
  d2 marker fsquare err 30%
    errwidth .1
end graph

```



dn herr d5 herrwidth *width-exp* dn herrleft *nn%* errright d4

These commands are identical to the error bar commands above except that they will draw bars in the horizontal plane.

dn key "Dataset title"

If a dataset is given a title like this a key will be drawn. Use the key command (below, after `hscale`) to set the size and position of the key. Use the key module (Chapter 4) to draw more complex keys.

dn line [impulses] [steps] [fsteps] [hist] [svg\_smooth]

This tells GLE to draw lines between the points of the dataset. By default GLE will not draw lines or markers, this is often the reason for a blank graph.

If a dataset has missing values GLE will not draw a line to the next real value, which leaves a gap in the curve. To avoid this behavior simply use the `nomiss` qualifier on the `dn` command used to define the line. This simply throws away missing values so that lines are drawn from the last real value to the next real value.

The option `svg_smooth` performs a Savitski Goulay smoothing on the data.

The options `impulses`, `steps`, `fsteps`, and `hist` draw lines as shown in Figure 3.1.

- **impulses**: connects each point with the xaxis.
- **steps**: connects consecutive points with two line segments: the first from (x1,y1) to (x2,y1) and the second from (x2,y1) to (x2,y2).
- **fsteps**: connects consecutive points with two line segments: the first from (x1,y1) to (x1,y2) and the second from (x1,y2) to (x2,y2).
- **hist**: useful for drawing histograms: assumes that each point is the center of a bin of the histogram.

dn lstyle *line-style* *lwidth* *line-width* *color* *col*

These qualifiers are all fairly self explanatory. See the `lstyle` command in Chapter 2 for details of specifying line styles.

dn marker *marker-name* [*msize* *marker-size*] [*mdata* *dn*]

Specifies the marker to be used for the dataset. There is a set of pre-defined markers (refer to Appendix A.1 for a list) which can be specified by name (e.g., `circle`, `square`, `triangle`, `diamond`, `cross`, ...). Markers can also be a user-defined subroutine (See the `define marker`

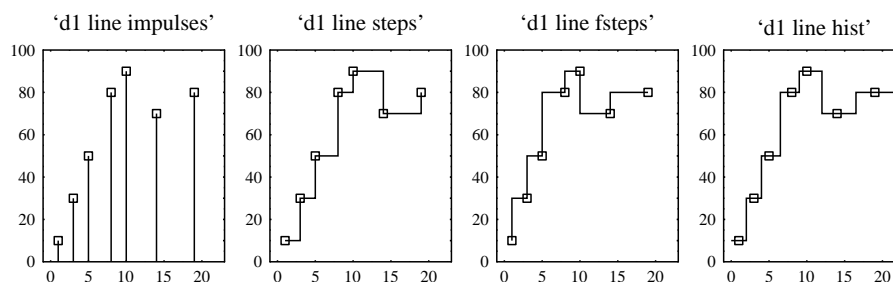


Figure 3.1: The impulses, steps, fsteps, and hist options of the line command.

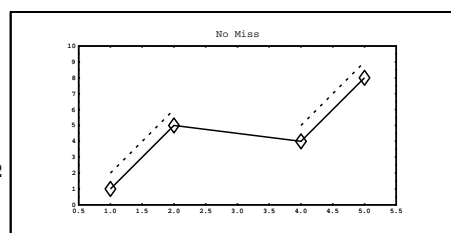
command in Chapter 2). The `mdata` option allows a secondary dataset to be defined which will be used to pass another parameter to the marker subroutine, this allows each marker to be drawn at a different angle, size or colour.

The `msize` qualifier sets the marker size for that dataset. The size is a character height in cm, so that the actual size of the markers will be about 0.7 of this value.

#### dn nomiss

If a dataset has missing values, GLE will not draw a line to the next real value, which leaves a gap in the curve. To avoid this behavior simply use the `nomiss` qualifier on the `dn` command used to define the line. This simply ignores missing values.

```
size 6 3
begin graph
  title "No Miss"
  data tut3.dat
  yaxis min 0
  d2 nomiss lstyle 1
  marker diamond msize .2
end graph
```



#### dn smooth — smoothm

This will make GLE draw a smoothed line through the points. A third degree polynomial is fitted piecewise to the given points.

The `smoothm` alternative will work for multi valued functions, i.e., functions which have more than one  $y$  value for each  $x$  value.

#### dn xmin $x$ -low xmax $x$ -high ymin $y$ -low ymax $y$ -high

These commands map the dataset onto the graph's boundaries. The data will be drawn as if the  $X$  axis was labelled from  $x$ -low to  $x$ -high (regardless of how the axis is actually labelled). A point in the dataset at  $X = x$ -low will appear on the left hand edge of the graph.

#### fullsize

This is equivalent to `vscale 1`, `hscale 1`, `noborder`. It makes the graph size command specify the size and position of the axes instead of the size of the outside border.

#### hscale exp

Scales the length of the yaxis. See `vscale`. The default value is 0.7.

key pos *tl* nobox hei *exp* offset *xexp* *yexp*

This command allows the features of a key to be specified. The **pos** qualifier sets the position of the key. E.g., **tl**=topleft, **br**=bottomright, etc.

let *ds* = *exp* [ from *low* to *high* step *exp* ]

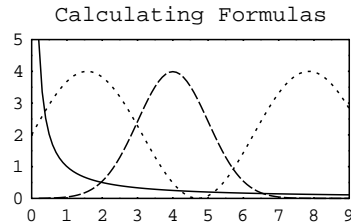
This command defines a new dataset as the result of an expression on the variable *x* over a range of values. It also allows the use of other datasets. E.g., to generate an average of two datasets:

```
data aa.dat d1 d2
let d3 = d1+d2/2
```

Or to generate data from scratch:

```
let d1 = sin(x)+log(x) from 1 to 100 step 1
```

```
2pi = 2*3.14
begin graph
  size 6 3
  let d1 = 1/x from 0.2 to 10
  let d2 = sin(x)*2+2 from 0.2 to 10
  let d3 = 10*(1/sqrt(2pi))*
    exp(-2*(sqr(x-4)/sqr(2)))
    from 0.2 to 10 step .1
  dn line
  d2 lstyle 2
  d3 lstyle 3
end graph
```



If the xaxis is a LOG axis then the **step** option is read as the number of steps to produce rather than the size of each step.

**NOTE: The spacing around the '=' sign and the lack of spaces inside the expression are necessary.**

let *dn* = [routine] *dm* [options]

GLE includes several fitting routines that allow an equation to be fit to a data series. These routines can be included in a 'let' expression as shown above, where *dn* will contain results of fitting *routine* to the data in *dm*, and the [options] control the limits to which the data in *dn* extends.

The following routines are available :

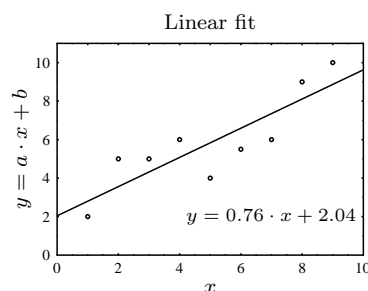
- **linfit**: fits the data in *dm* to the straight line equation  $y = m \cdot x + b$ .
- **logfit**: fits the data in *dm* to the equation  $y = a \cdot \exp(b \cdot x)$ .
- **log10fit**: fits the data in *dm* to the equation  $y = a \cdot 10^{b \cdot x}$ .
- **powxfits**: fits the data in *dm* to the equation  $y = a \cdot x^b$ .

The following options are available :

- **limit\_data\_x** The range of the data in *dn* extends from the minimum *x* value in *dm* to the maximum *x* value in *dm*.
- **limit\_data\_y** The range of the data in *dn* extends from the *x* value of the minimum *y* value in *dm* to the *x* value of the maximum *y* value in *dm*.

- **limit\_data** The range of the data in *dn* extends from the greater of the *x* value of the minimum *y* value or the minimum *x* value in *dm* to the greater of the *x* value of the maximum *y* value or the maximum *x* value in *dm*.
- **from *xmin* to *xmax*** The range of the data in *dn* extends from the *xmin* to *xmax* as specified by the user.

```
begin graph
...
data fitlin.dat
let d2 = linfit d1 from 0 to 10
d1 marker circle
d2 line
end graph
```



#### nobox

This removes the outer border from the graph.

#### size *x y*

Defines the size of the graph in cm. This is the size of the outside box of a graph. The default size of the axes of the graph will be 70% of this, (see *vscale* and *hscale*). This command is required.

#### title "*title*" [*hei ch-hei*] [*color col*] [*font font*] [*dist cm*]

This command gives the graph a centred title. The list of optional keywords specifies features of it. The *dist* command is used for moving the title up or down.

#### vscale *exp*

This sets the width of the axis relative to the width of the graph. For example with a 10cm wide graph and a *vscale* of .6 the x axis would be 6cm long. A setting of 1.0 makes the xaxis the same length as the width of the graph, which is useful for positioning some graphs. The default value is 0.7.

#### x2labels on

This command 'activates' the numbering of the x2axis. There is a corresponding command 'y2axis on' which will activate y2axis numbering.

#### xaxis — yaxis — x2axis — y2axis

A graph is considered to have four axes: The normal xaxis and yaxis as well as the top axis (x2axis) and the right axis (y2axis).

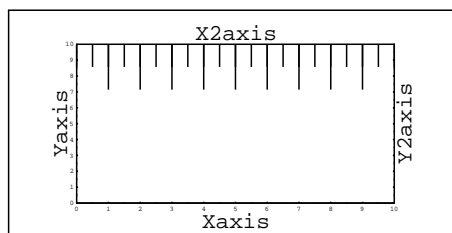
Any command defining an xaxis setting will also define that setting for the x2axis.

The secondary axes x2 and y2 can be modified individually by starting the axis command with the name of that axis. E.g.,

```

begin graph
  size 6 3
  xtitle "Xaxis" hei .3
  ytitle "Yaxis" hei .3
  x2title "X2axis" hei .3
  y2title "Y2axis" hei .3
  x2ticks length .6
end graph

```



**xaxis base** *exp-cm*

Scale the axis font and ticks by *exp-cm*.

**xaxis color** *col* **font** *font-name* **hei** *exp-cm* **lwidth** *exp-cm*

These axis qualifiers affect the colour, lstyle, lwidth, and font used for drawing the xaxis (and the x2axis). These can be overridden with more specific commands. E.g., 'xticks color blue' would override the axis colour when drawing the ticks. The subticks would also be blue as they pick up tick settings by default.

**xaxis dsubticks** *sub-distance*

See xaxis nticks below.

**xaxis format** *format-string*

Specifies the number format for the labels. See the documentation of **format\$** on page 21 for a description of the syntax. Example:

```
xaxis format "fix 1"
```

**xaxis grid**

This command makes the xaxis ticks long enough to reach the x2axis and the yaxis ticks long enough to reach the y2axis. When used with both the x and y axes this produces a grid over the graph. Use the **xticks lstyle** command to create a faint grid.

**xaxis log**

Draws the axis in logarithmic style, and scales the data logarithmically to match (on the x2axis or y2axis it does not affect the data, only the way the ticks and labelling are drawn)

Be aware that a straight line should become curved when drawn on a log graph. This will only happen if you have enough points or have used the **smooth** option.

**xaxis min** *low* **max** *high*

Sets the minimum and maximum values on the xaxis. This will determine both the labelling of the axis and the default mapping of data onto the graph. To change the mapping see the dataset **dn** commands **xmin**, **ymin**, **xmax**, and **ymax**.

**xaxis nofirst nolast**

These two switches simply remove the first or last (or both) labels from the graph. This is useful when the first labels on the x and y axis are too close to each other.

**xaxis nticks** *number* **dticks** *distance* **dsubticks** *distance*

**nticks** specifies the number of ticks along the axis. **dticks** specifies the distance between ticks and **dsubticks** specifies the distance between subticks. For example, to get one subtick between every main tick with main ticks 3cm apart, simply specify **dsubticks 1.5**. Alternatively, one can also use **nsubticks**.

By default ticks are drawn on the inside of the graph. To draw them on the outside use the command:

```
xticks length -.2
yticks length -.2
```

`xaxis off`

Turns the whole axis off — labels, ticks, subticks and line. Often the `x2axis` and `y2axis` are not required, they could be turned off with the following commands:

```
x2axis off
y2axis off
```

`xaxis shift cm-exp`

This moves the labelling to the left or right, which is useful when the label refers to the data between the two values.

`xlabels font font-name hei char-hei color col`

This command controls the appearance of the axis labels but not the axis title.

`xnames "name" "name" ...`

This command replaces the numeric labelling with absolutely anything. Given data consisting of five measurements, taken from Monday to Friday, one per day then

```
xaxis min 0 max 6 dticks 1
xnames "" "Mon" "Tue" "Wed" "Thu" "Fri" ""
```

would give the desired result. Note it is essential to define a specific axis minimum, maximum, dticks, etc., otherwise the labels may not correspond to the data.

If there isn't enough room on the line for all the names then simply use an extra `xnames` command.

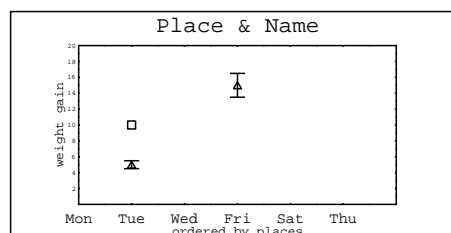
`xplaces pos1 pos2 pos3 ...`

This is similar to the `xnames` command but it specifies a list of points which should be labelled. This allows labelling which isn't equally spaced. The above example with days of the week could be written like this:

```
xplaces 1 2 5
xplaces 7
xnames "Mon" "Tue" "Fri" "Sun"
```

If there isn't enough room on the line for all the names then simply use an extra `xplaces` command.

```
begin graph
  size 6 3
  xtitle "ordered by places"
  ytitle "weight gain"
  title "Names & Places"
  xaxis min 0 max 6 dticks 1
  xplaces 0 1 2 5 3 4
  xnames "Mon" "Tue" "Wed" "Thu" "Fri" "Sat"
  yaxis min 0 max 20 nofirst
  data test.dat
  d1 marker square color blue
  d2 marker triangle color red
  d2 err 10% errwidth .5
end graph
```



`xside color col lwidth line-width off`

This command controls the appearance of the axis line, i.e. the line to which the ticks are attached.

`xsubticks lstyle num lwidth exp length exp off`

This command gives fine control of the appearance of the axis subticks.

`xticks lstyle num lwidth exp length exp off`

This command gives fine control of the appearance of the axis ticks. Note: To get ticks on the outside of the graph, i.e. pointing outwards, specify a negative tick length:

```
xticks length -.2
yticks length -.2
```

`xtitle "title" [hei ch-hei] [color col] [font font] [dist cm]`

This command gives the axis a centered title. The list of optional keywords specify features of it. The `dist` command is used for moving the title up or down.

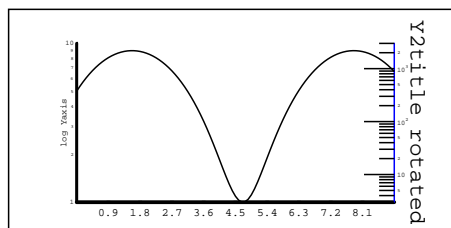
`xaxis negate`

This is reversed the numbering on the y axis. For use with measurements below ground, where you want zero at the top and positive numbers below the zero.

`y2title "text-string" [rotate]`

By default the `y2title` is written vertically upwards. The optional `rotate` keyword changes this direction to downwards. The `rotate` option is specific to the `y2title` command.

```
begin graph
  xaxis min 0 max 9 nofirst nola
  xaxis hei .6 nticks 10 dsubticks .2
  xaxis lwidth .2 color red
  ytitle "log Yaxis"
  yaxis log min 1 max 10 lwidth .1
  y2axis min 3 max 3000
  y2side color blue lwidth 0
  y2ticks length .9 lwidth 0
  y2title "Y2title rotated" hei .3 rotate
  x2axis off
  y2labels on
  let d1 = sin(x)*4+5 from 0 to 9
  dn line
end graph
```



### 3.3 Bar Graphs

Drawing a bar graph is a subcommand of the normal graph module. This allows bar and line graphs to be mixed. The bar command is quite complex as it allows a great deal of flexibility. The same command allows stacked, overlapping and grouped bars.

For stacked bars use separate bar commands as in the first example below:

```
bar d1 fill black
bar d2 from d1 fill grey10
```

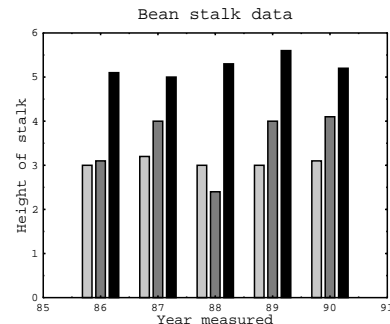
For grouped bars put all the datasets in a list on a single bar command:

```
bar d1,d2,d3 fill grey10,grey40,black
```

```

begin graph
  nobox
  size 6.5 5
  title "Bean stalk data"
  xtitle "Year measured"
  ytitle "Height of stalk"
  xaxis min 85 max 91 dticks 1
  yaxis min 0 max 6
  data gc_bean.dat
  bar d1,d2,d3
  fill grey10,grey40,black
end graph

```



**bar *dx*,... dist *spacing***

Specifies the distance between bars in dataset(s) *dx*,.... The distance is measured from the left hand side of one bar to the left hand side of the next bar. A distance of less than the width of a bar results in the bars overlapping.

**bar *dx*,... from *dy*,...**

This sets the starting point of each bar in datasets *dx*,... to be at the value in datasets *dy*,..., and is used for creating stacked bar charts. Each layer of the bar chart is created with an additional bar command.

```

bar d1,d2
bar d3,d4 from d1,d2
bar d5,d6 from d3,d4

```

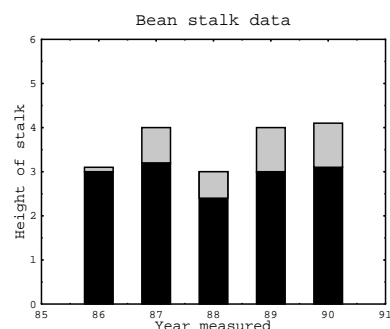
Note 1: It is important that the values in *d3* and *d4* are greater than the values in *d1* and *d2*.

Note 2: Data files for stacked bar graphs should not have missing values, replace the \* character with the number on its left in the data file.

```

begin graph
  nobox
  size 6.5 5
  title "Bean stalk data"
  xtitle "Year measured"
  ytitle "Height of stalk"
  xaxis min 85 max 91 dticks 1
  yaxis min 0 max 6
  data gc_bean.dat
  bar d1 fill black
  bar d2 from d1 fill grey10
end graph

```



**bar *dn*,... width *xunits*,... fill *col*,... color *col*,...**

The rest of the bar qualifiers are fairly self explanatory. When several datasets are specified, separate them with commas (with no spaces between commas).

```

bar d1,d2 width 0.2 dist 0.2 fill grey10,grey20 color red,green

```

### 3.4 3D Bar Graphs

3d Bar graphs are now supported, the commands are:

```
bar d1,d2 3d .5 .3 side red,green notop
bar d3,d4 3d .5 .3 side red,green top black,white
```

Take note of comma's.

*bar dx,... 3d xoffset yoffset side color list top color list [notop]*

*3d xoffset yoffset*

Specifies the x and y vector used to draw the receding lines, they are defined as fractions of the width of the bar. A negative xoffset will draw the 3d bar on the left side of the bar instead of the right hand side.

*side color list*

The color of the side of each of the bars in the group.

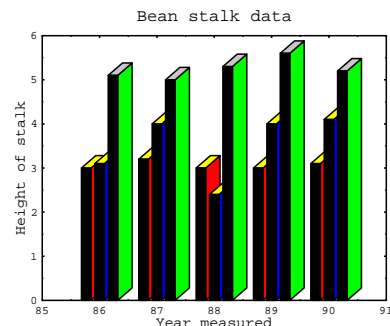
*top color list*

The color of the top part of the bar

*notop*

Turns off the top part of the bar, use this if you have a stacked bar graph so you only need sides on the lower parts of each stack.

```
begin graph
  nobox
  size 6.5 5
  title "Bean stalk data"
  xtitle "Year measured"
  ytitle "Height of stalk"
  xaxis min 85 max 91 dticks 1
  yaxis min 0 max 6
  data gc_bean.dat
  bar d1,d2,d3 3d 1.5 1.3
    side red,blue,green
    top yellow,yellow,greyl0
end graph
```



Note: You won't see the color of the side or top on the pc screen.

### 3.5 Filling Between Lines

*fill x1,d3 color green xmin val xmax val*

Fills between the xaxis and a dataset, use the optional xmin, xmax, ymin, ymax qualifiers to clip the filling to a smaller region

*fill d4,x2 color blue ymin val ymax val*

This command fills from a dataset to the x2axis.

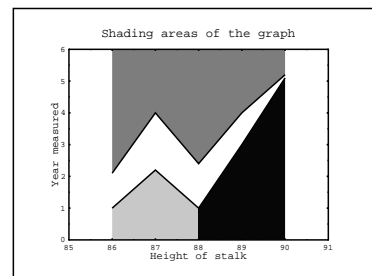
`fill d3,d4 color green xmin val xmax val`

This command fills between two datasets.

`fill d4 color green xmin val xmax val`

This command treats the dataset as a polygon and fills it. The dataset should be a closed polygon.

```
fill d2,x2 color grey40
fill x1,d1 color grey10 xmin 85 xmax 88
fill x1,d1 color grey90 xmin 88 xmax 91
```



## 3.6 Notes on Drawing Graphs

### 3.6.1 Importance of Order

Most of the graph commands can appear in any order, but in some cases order is significant.

As some let commands operate on data which has been read into datasets, the `data` commands should precede the let commands.

The wildcard `dn` command should appear before specific `d1` commands which it will override.

By default `xaxis` commands also change the `x2axis`, and `xlabels` commands also change `x2labels`, so to specify different settings for the `x` and `x2` axes, put the `x2` settings after the `x` settings.

```
begin graph
  size 10 10
  data a.dat
  let d2 = d1*3
  dn marker square lstyle 3    ! sets d1 and d2
  d2 marker dot
  xaxis color green
  xticks color blue
  x2axis color black
end graph
```

### 3.6.2 Line Width

When scaling a graph up or down for publication the default line width may need changing. To do this simply specify a `set lwidth` command before beginning the graph.

```
size 10 10
set lwidth .1
begin graph
  ...
end graph
```

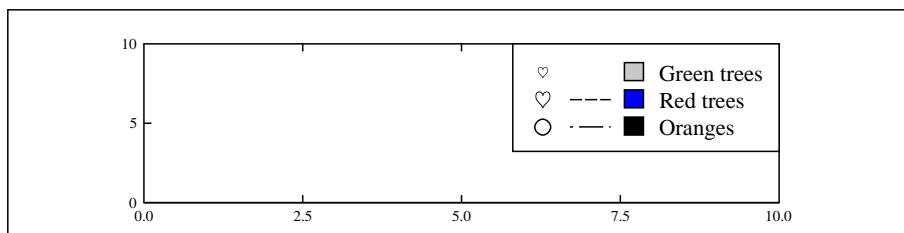
## Chapter 4

# The Key Module

This module is used for drawing keys for graphs. It is completely separate from the graph module.

Specify a position for the key, the size of the lettering and then for each dataset specify marker, color, lstyle and fill pattern:

```
begin graph
...
end graph
begin key
  hei .5
  position tr
  text "Green trees" marker heart msize .2 fill grey10
  text "Red trees" lstyle 3 marker heart fill blue
  text "Oranges" marker circle lstyle 6 fill black
end key
```



All commands to do with a particular line of the key module **MUST** appear on the same line.

### 4.1 Key commands

`offset x-exp y-exp [absolute]`

Specifies the offset in cm from the current point (unless the option absolute is given) to

the anchor point of the key. By default the anchor point is the bottom left hand corner of the key.

**position** *justify-exp*

Specifies the position of the key on the graph or the anchor point of the key (if used in combination with **offset**).

**text** *str-exp*

The text which will be displayed on the end of the line.

**lstyle** *style-num*

The line style which will be used for the short line drawn in the key.

**marker** *marker-name*

The marker which will be used for that line of the key.

**msize** *exp*

Specifies the size of the markers in cm.

**mscale** *exp*

Specifies how much to scale the size of the marker. E.g., 0.5 would produce a marker half as big as normal.

**color** *color-name*

The colour of the text, line and marker.

**hei** *cm-exp*

This sets the height of the text used to draw the key. The key will change in size to fit around the text. If you omit this command the current font size is used.

**fill** *fill-pattern*

The fill pattern used in that line of the key.

**separator** [*lstyle exp*]

Use this command to divide the key into several columns.

## 4.2 Key syntax

The syntax of the KEY module is rather confusing.

The commands which relate to global features of the key:

**offset, nobox, hei, pos**

Can appear anywhere except on the start of a line which defines a new dataset, e.g.:

```
begin key
  offset 2 3 hei 2
  text "abc" nobox
  nobox text "abc" (this line is not valid)
  ...
end key
```

The commands which relate to a single dataset (or line of the key):

```
text, marker, msize, mscale, color, fill, lstyle, line, lwidth
```

Must appear together on a line, but any one of them can start the line, e.g.:

```
begin key
  marker circle text "abc" lstyle 2
  lstyle 2 marker square
end key
```

So you usually put the global commands first, and then one line for each line in the key, e.g.:

```
begin key
  nobox pos tl hei .5 (global commands)
  marker circle text "The circles"
  marker square lstyle 1 text "The squares"
end key
```

A key with more than one column can be defined using the keyword **separator** (the columns can be separated with a line using the **lstyle** option).

```
begin key
  marker circle text "method 1"
  separator
  marker square text "method 2"
end key
```

There are two ways to specify the position of the key.

1. The first one is relative to the graph. Use the *position* command to specify the position on the graph, e.g., *tl* = top left. See **set just** for a list of justify settings.
2. The second one is absolute positioning. Use the *offset* command **together** with the *position* command. The following example positions the key at the bottom-center of the output.

```
amove 0 0
begin key
  offset pagewidth()/2 0.01
  position bc
  ...
```



## Chapter 5

# Advanced features

This chapter covers the advanced features of GLE.

### 5.1 Colour

Internally GLE treats color and fill identically, they are simply an intensity of Red, Green and Blue. Each of the predefined color names (yellow, grey20, orange, red) simply define the ratio of red, green and blue.

There are two ways to use variables to show color, one is for shades of grey:

```
for i = 0 to 10
  box 3 .2 fill (i/10)
  rmove 0 .2
next i
```

The other is for passing a color **name** as a variable:

```
sub stick c$
  box .2 2 fill c$
end sub
@stick "green"
```

A color can also be defined based on its RGB values with the CVTRGB primitive.

```
mycolor$ = "CVTRGB(38/255,38/255,134/255)"
```

Remember a fill pattern completely obscures what is behind it, so the following command would produce a box with a shadow:

```
amove 4 4
box 3 2 fill grey10
```

```

rmove -.1 .1
box 3 2 fill white
rmove .4 .4
text hellow

```

## 5.2 Diagrams, Joining Named Objects

To draw lines between boxes which contain text, first name each box as it is drawn and then use the join command to draw the lines between the boxes.

```

box 2 3 fill blue name square
amove 5 5
begin box add .1 name titlebox
    text Title
end box
join square.tr -> titlebox.bc

```

These commands draw a line from the “Top Right” of the square to the “Bottom Centre” of the titlebox, with an arrow at the titlebox end.

```
join square - titlebox
```

would draw a line from the centre of the square to the centre of the titlebox but clipped correctly at the edges of both boxes.

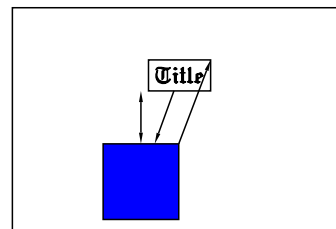
```
join square.tc <-> titlebox.v
```

would draw a vertical line from the top centre of the square to the titlebox with arrows at both ends.

```

set hei .3 font plge
amove 1.2 .2
box 1 1 fill blue name square
amove 1.9 2
begin box add .1 name titlebox
    text Title
end box
join square.tr -> titlebox.tr
join square <- titlebox
join square.tc <-> titlebox.v

```



Named points on each box:

```

.bl Bottom left
.bc Bottom centre
.br Bottom right
.cr Centre right
.tr Top right
.tc Top centre
.tl Top left
.cl Centre left

```

```
.v Vertical line
.h Horizontal line
.cc Centre centre
.ci Circle clipping (for drawing lines to a circle)
```

To draw lines to a given point, simply move there and save that point as a named object.

```
rmove 2 3
save apoint
join apoint - square
```

## 5.3 $\text{\LaTeX}$ Interface

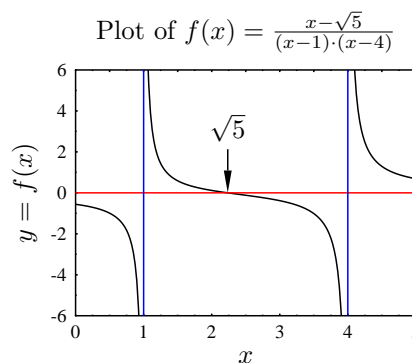
### 5.3.1 Example

GLE files can include arbitrary  $\text{\LaTeX}$  expressions using the  $\text{\LaTeX}$  interface. There are two ways to include a  $\text{\LaTeX}$  expression. The first one is by using the ‘`tex`’ primitive. The second one is by using the ‘`\tex{}`’ macro in a string.

```
begin graph
...
title "\tex{Plot of $f(x) = \frac{x-\sqrt{5}}{(x-1)\cdot(x-4)}$}"
xtitle "\tex{$x$}"
ytitle "\tex{$y = f(x)$}"
...
end graph

amove xg(sqrt(5)) yg(2.5)
begin box add 0.1 name sq5b nobox
set just bc
tex "$\sqrt{5}$"
end box

amove pointx(sq5b.bc) pointy(sq5b.bc)
aline xg(sqrt(5)) yg(0) arrow end
```



$\text{\LaTeX}$  expressions are drawn on top of all other graphics and cannot be clipped (Section 5.4).

$\text{\LaTeX}$  expressions respect the ‘`just`’ setting, but ignore the ‘`hei`’ setting. As a result, the font sizes in your graphics will be exactly the same as in your main document. To obtain different font sizes, use the font size primitives provided by  $\text{\LaTeX}$  (e.g., `\large`, ...). In the future, a setting ‘`scaletexfonts`’ will be added to control scaling of the fonts in  $\text{\LaTeX}$  expressions. If this setting is true, GLE will respect the value of ‘`hei`’.

### 5.3.2 Import in a TeX document

If your source file contains  $\text{\LaTeX}$  expressions then GLE will create besides the usual .eps or .pdf file also an .inc file. This .inc file can be imported in a  $\text{\LaTeX}$  document as follows.

```
\input{myfile.inc}
```

The .inc file tells `latex` (or `pdflatex`) to include the .eps/.pdf output file created by GLE. It also includes  $\text{\TeX}$  draw commands for drawing the  $\text{\LaTeX}$  expressions on top of the GLE output. Note that the .eps/.pdf file created by GLE does not include these (you can check this by viewing it with GhostView).

To be able to include .inc files, the following must be included in the preamble of your  $\text{\LaTeX}$  document.

```
\usepackage[dvips]{graphics}
\usepackage{color}
```

If you use `pdflatex`, then the `dvips` option of the `graphics` package should be replaced by `pdfTeX`.

If you place your .gle files in a subdirectory of the directory where your  $\text{\LaTeX}$  document resides, the .inc file created by GLE should include the path to this subdirectory in the '`\includegraphics`' primitive it uses for including the .eps/.pdf file generated by GLE. To add this path, use the '`texincprefix`' command line option of GLE. For example, if your GLE files are in a subdirectory called 'plots' then one should run GLE as follows.

```
gle -texincprefix "plots/" -d eps myfile.gle
```

GLE can color and rotate  $\text{\LaTeX}$  expressions (use '`set color`' and '`begin rotate`'). Note however that '`xdvi`' does not support these effects, so you will not be able to see them if you use this viewer. In the final PostScript or PDF output, they will of course be displayed correctly.

### 5.3.3 Viewing the Output

As should be clear from the previous section, the output .eps/.pdf file created by GLE does not include the  $\text{\LaTeX}$  expressions. These are drawn by  $\text{\LaTeX}$  itself when the .inc file is included in a document.

To help editing a script, GLE provides an extra output format called '`tEps`', which is similar to .eps but it also includes the  $\text{\LaTeX}$  expressions. GLE automatically runs  $\text{\LaTeX}$  to create this output format. A '`tEps`' file can be created as follows.

```
gle -d tEps myfile.gle
```

Now, `myfile.tEps` will contain both the GLE graphics and the  $\text{\LaTeX}$  expressions and it can be viewed easily with GhostView.

Note that it is also possible to include `myfile.tEps` in a  $\text{\LaTeX}$  document with the '`\includegraphics`' primitive. This will however result in a larger output as is the case if one includes the .inc file constructed by GLE.

### 5.3.4 The .gle Directory

If your source includes  $\text{\LaTeX}$  expressions, then GLE will construct a subdirectory called ‘.gle’ for storing temporary files (e.g., used for measuring the printed size of the  $\text{\LaTeX}$  expressions). After you are finished you can safely delete the .gle directory. GLE will recreate it automatically if required.

## 5.4 Filling, Stroking and Clipping Paths

It is possible to set up arbitrary clipping regions. To do this draw a shape and make it into a path by putting a `begin path clip ... end path`, around it. Then draw the things to be clipped by that region. To clear a clipping path surround the whole section of GLE commands with `begin clip ... end clip`

Characters can be used to make up clipping paths, but only the PostScript fonts will currently work for this purpose.

```
size 10 5
begin clip      ! Save current clipping path
  begin path clip stroke ! Define new clipping region
    amove 2 2
    box 3 3
    amove 6 2
    box 3 3
  end path
  amove 2 2
  set hei 3
  text Here is clipped text
end clip        ! Restore original clipping path
```



## 5.5 Using Variables

GLE has two types of variables, floating point and string. String variables always end with a dollar sign. A string variable contains text like “Hello this is text”, a floating point variable can only contain numbers like 1234.234.

```
name$ = "Joe"
height = 6.5      ! Height of person
shoe = .05        ! shoe adds to height of person
amove 1 1
box .2 height+shoe
write name$
```

## 5.6 Programming Loops

The simple way to draw a  $6 \times 8$  grid would be to use a whole mass of line commands:

```
amove 0 0
rline 0 8
amove 1 0
rline 1 8
...
amove 6 0
rline 6 8
```

this would be laborious to type in, and would become impossible to manage with several grids. By using a simple loop this can be avoided:

```
for x = 0 to 6
  amove x 0
  rline x 8
next x
for y = 0 to 8
  amove 0 y
  rline 6 y
next y
```

To draw lots of grids all of different dimensions a subroutine can be defined and then used again and again:

```
! define the subroutine
sub grid nx ny
gsave
begin origin
for x = 0 to nx
  amove x 0
  aline x ny
next x
```

```

for y = 0 to ny
  amove 0 y
  aline nx y
next y
end origin
end sub
! now draw the grids wherever
amove 2 4
@grid 6 8
amove 2 2
@grid 9 5

```

Now the main GLE file will be much easier to modify particularly if the subroutine definition is moved into a separate file:

```

size 10 10
include griddef.gle
amove 2 4
@grid 2 4
amove 2 2
@grid 9 5

```

## 5.7 Extended I/O Functions

Some standard I/O function and macros are available:

fopen, fclose, fread, freadln, fwrite, fwriteln  
e.g.:

```

fopen "file.dat" inchan read
fopen "file.out" outchan write
until feof(inchan)
  fread inchan x y z
  aline x y
  rline x z
  fwriteln outchan x*2 "y =" y
next
fclose inchan
fclose outchan

```

## 5.8 Device dependend Control

A built in function which returns a string describing the device is available.  
e.g. DEVICE\$() = "HARDCOPY, PS,"  
on the postscript driver.

This can be used to use particular fonts etc on appropriate devices. E.g.:

```
if pos(device$(),"PS",1)>0 then
  set font psncsb
end if
```

## Chapter 6

# Surface and Contour Plots

### 6.1 Surface Primitives

#### 6.1.1 Overview

SURFace plots 3 dimensional data using a wire frame with hidden line removal.

The simplest SURFACE code would look like this.

```
begin surface
DATA myfile.z 5 5
end surface
```

More help is available on the following topics:

```
SIZE x y
CUBE [OFF] [XLEN v] [YLEN v] [ZLEN v] [NOFRONT] [LSTYLE l] [COLOR c]
DATA myfile.z [XSAMPLE n1] [YSAMPLE n2] [SAMPLE n3] [NX n1] [NY n2]
HARRAY n
XLines — YLines [OFF]
XAxis — YAxis — ZAxis [MIN v] [MAX v] [STEP v] [COLOR c] [LSTYLE l] [HEI v] [OFF]
XTITLE — YTITLE — ZTITLE "title" [DIST v] [COLOR c] [HEI v]
TITLE "Main title" [DIST v] [COLOR c] [HEI v]
ROTATE  $\theta$   $\phi$  x
VIEW x y p
TOP — UNDERNEATH [OFF] [LSTYLE n] [COLOR c]
BACK [ZSTEP v] [YSTEP v] [LSTYLE l] [COLOR c] [NOHIDDEN]
BASE [XSTEP v] [YSTEP v] [LSTYLE l] [COLOR c] [NOHIDDEN]
RIGHT [ZSTEP v] [XSTEP v] [LSTYLE l] [COLOR c] [NOHIDDEN]
SKIRT ON
POINTS myfile.dat
MARKER circle [HEI v] [COLOR c]
```

DROPLINES — RISELINES [COLOR *c*] [LSTYLE *n*]  
 ZCLIP [MIN *v1*] [MAX *v2*]

### 6.1.2 Surface Commands

size *x y*

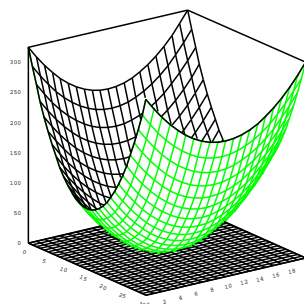
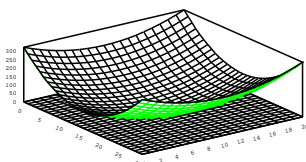
Specifies the size in cm to draw the surface. The 3d cube will fit inside this box. The default is 18cm x 18cm e.g. `SIZE 10 10`

cube [OFF] [XLEN *v*] [YLEN *v*] [ZLEN *v*] [NOFRONT] [LSTYLE *l*] [COLOR *c*]

Surface is drawing a 3d cube.

OFF	Stops GLE from drawing the cube.
XLEN	The length of the cubes x dimension in cm.
NOFRONT	Removes the front three lines of the cube.
LSTYLE	Sets the line style to use drawing the cube.
COLOR	Sets the color of lines to use drawing the cube.

```
begin surface
size 10 10
data jack.z
cube zlen 5
end surface
```



DATA it myfile.z [XSAMPLE *n1*] [YSAMPLE *n2*] [SAMPLE *n3*] [NX *n1*] [NY *n2*]

Loads a file of Z values in. The NX and NY dimensions are optional, normally the dimensions of the data will be defined on the first line of the data file. e.g.

```
! NX 10 NY 20 XMIN 1 XMAX 10 YMIN 1 YMAX 20
1 2 42 4 5 2 31 4 3 2 4
1 2 42 4 5 2 31 4 3 2 4 etc...
```

```
y1,x1, y1,x2 ... y1,xn
y2,x1, y2,x2 ... y2,xn
.
.
.
yn,x1, yn,x2 ... yn,xn
```

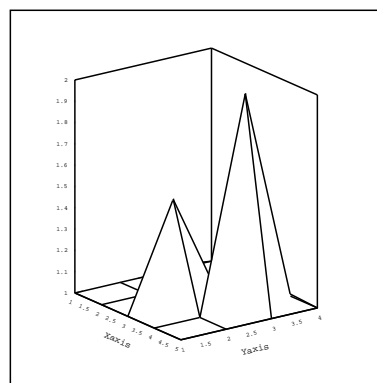
Data files can be created using LETZ or FITZ. LETZ will create a data file from an x,y function. FITZ will create a data file from a list of x,y,z data points.

**XSAMPLE** Tells surface to only read every n'th data point from the data file. This speeds things up while you are messing around.

**YSAMPLE** Tells surface to only read every n'th line from the data file. (see also POINTS)

**SAMPLE** Sets both XSAMPLE and YSAMPLE

```
begin surface
  size 5 5
  xtitle "Xaxis"
  ytitle "Yaxis"
  data surf1.z
end surface
```



**harray *n***

The hidden line removal is accomplished with the help of an array of heights which record the current horizon, the quality of the output is proportional to the width of this array. (also the speed of output)

To get good quality you may want to increase this from the default of about 900 to 2 or 3 thousand. e.g. **HARRAY 2000**

**xlines off**

Stops SURF from drawing lines of constant X.

**Y LINES OFF**

Stops SURF from drawing lines of constant Y.

**XAXIS [MIN *v*] [MAX *v*] [STEP *v*] [COLOR *c*] [LSTYLE *l*] [HEI *v*] [OFF]**

**ZAXIS [MIN *v*] [MAX *v*] [STEP *v*] [COLOR *c*] [LSTYLE *l*] [HEI *v*] [OFF]**

**YAXIS [MIN *v*] [MAX *v*] [STEP *v*] [COLOR *c*] [LSTYLE *l*] [HEI *v*] [OFF]**

**MIN,MAX** Set the range used for labelling the axis.

**STEP** The distance between labels on the axis.

**COLOR** The color of the axis ticks and labels.

**LSTYLE** The line style used for drawing the ticks.

**TICKLEN** The length of the ticks.

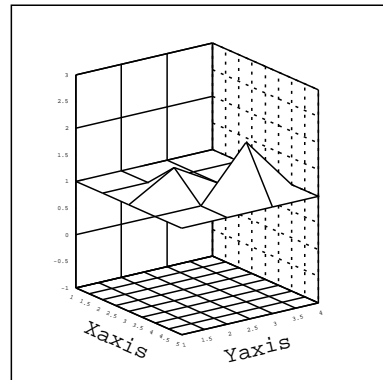
**HEI** The height of text used for labelling.

**OFF** Stops GLE from drawing the axis.

```

begin surface
  size 5 5
  data surf1.z
  zaxis min -1 max 3
  base xstep .5 ystep .5
  back ystep 1 zstep 1
  right xstep .5 zstep .5 lstyle 2
  xtitle "Xaxis" hei .3
  ytitle "Yaxis" hei .3
end surface

```



XTITLE "X title" [DIST *v*] [COLOR *c*] [HEI *v*]

YTITLE "Y title" [DIST *v*] [COLOR *c*] [HEI *v*]

ZTITLE "Z title" [DIST *v*] [COLOR *c*] [HEI *v*]

DIST Moves the title further away from the axis.  
 COLOR Sets the color of the title.  
 HEI Sets the hei in cm of the text used for the title.

TITLE "*Main title*" [DIST *v*] [COLOR *c*] [HEI *v*]

DIST Moves the title further away from the axis.  
 COLOR Sets the color of the title.  
 HEI Sets the hei in cm of the text used for the title.

ROTATE  $\theta$   $\phi$   $\times$

ROTATE 10 20 30

Imagine the unit cube is sitting on the front of your terminal screen, x along the bottom, y up the left hand side, and z coming towards you.

The first number (10) rotates the cube along the xaxis, ie hold the right hand side of the cube and rotate your hand clockwise 10 degrees.

The second number (20) rotates the cube along the yaxis, ie hold the top of the cube and rotate it 20 degrees clockwise.

The third number is currently ignored.

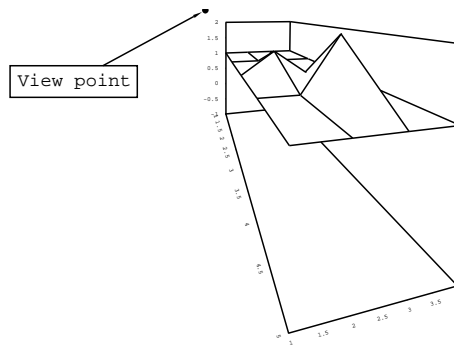
The default setting is 60 50 0.

VIEW *x y p*

Sets the perspective, this is where the cube gets smaller as the lines dissappear towards infinity.

*x* and *y* are the position of infinity on your screen. *p* is the degree of perspective, 0 = no perspective and with 1 the back edge of the box will be touching infinity. Good values are between 0 and 0.6

```
begin surface
  size 5 5
  data surf1.z
  zaxis min -1
  rotate 85 85 0
  view 0 5 .7
end surface
```



TOP [OFF] [LSTYLE *n*] [COLOR *c*]

Sets the features of the top of the surface. By default the top is on.

(see also UNDERNEATH, XLINES, YLINES)

UNDERNEATH [OFF] [LSTYLE *n*] [COLOR *c*]

Sets the features of the under side of the surface. By default the underneath is off.

(see also TOP, XLINES, YLINES)

BACK [ZSTEP *v*] [YSTEP *v*] [LSTYLE *l*] [COLOR *c*] [NOHIDDEN]

Draws a grid on the back face of the cube.

By default hidden lines are removed but NOHIDDEN will stop this from happening.

BASE [XSTEP *v*] [YSTEP *v*] [LSTYLE *l*] [COLOR *c*] [NOHIDDEN]

Draws a grid on the base of the cube.

By default hidden lines are removed but NOHIDDEN will stop this from happening.

RIGHT [ZSTEP *v*] [XSTEP *v*] [LSTYLE *l*] [COLOR *c*] [NOHIDDEN]

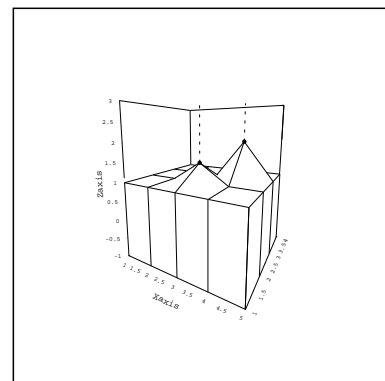
Draws a grid on the right face of the cube.

By default hidden lines are removed but NOHIDDEN will stop this from happening.

SKIRT ON

Draws a skirt from the edge of the surface to ZMIN.

```
begin surface
  size 5 5
  data surf1.z
  zaxis min -1 max 3
  xtitle "Xaxis"
  ztitle "Zaxis"
  points surf3.dat
  riselines lstyle 2
  marker fcircle
  skirt on
  rotate 80 30 0
  view 2.5 3 .6
end surface
```



POINTS *myfile.dat*

Reads in a data file which must have 3 columns (x,y,z)

This is then used for plotting markers and rise and drop lines.

MARKER *circle* [HEI *v* ] [COLOR *c* ]

Draws markers at the co-ordinates read from the POINTS file.

DROPLINES [COLOR *c* ] [LSTYLE *n* ]

Draws lines from the co-ordinates read from the POINTS file down to zmin.

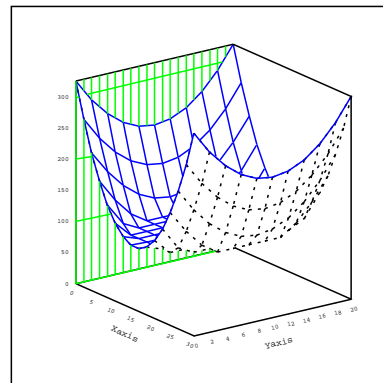
RISELINES [COLOR *c* ] [LSTYLE *n* ]

Draws lines from the co-ordinates read from the POINTS file up to zmax.

ZCLIP [MIN *v1* ] [MAX *v2* ]

ZCLIP goes through the Z array and sets any Z value smaller than MIN to *v1* and sets any value greater than MAX to *v2*.

```
begin surface
  size 5 5
  data jack.z sample 2
  back ystep 1 zstep 100
  color green
  xtitle "Xaxis"
  ytitle "Yaxis"
  top color blue
  zclip min 100
  zaxis min 0
  underneath on lstyle 2
end surface
```



## 6.2 Letz

LETZ generates a data file of z values given an expression in terms of x and y.

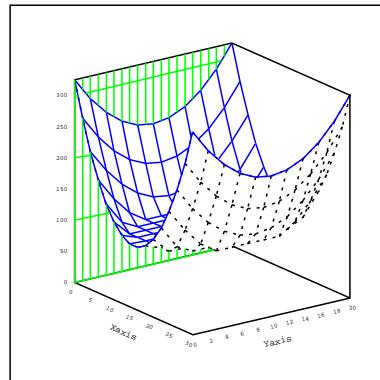
```
begin letz
  data "jack.z"
  z = x+sin(y^2)/pi+10.22
  x from 0 to 30 step 1
  y from 0 to 20 step 1
end letz
```

The file saddle.z now contains the required data.

```

begin surface
  size 5 5
  data jack.z sample 2
  back ystep 1 zstep 100
  color green
  xtitle "Xaxis"
  ytitle "Yaxis"
  top color blue
  zclip min 100
  zaxis min 0
  underneath on lstyle 2
end surface

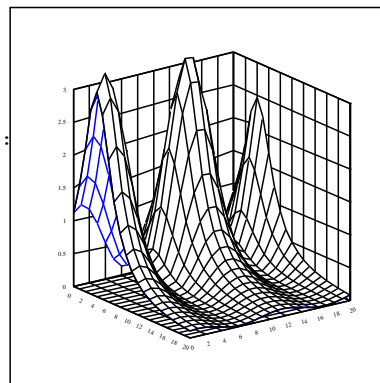
```



```

LETZ
File to store data in ?
      saddle.z
Enter xmin,xmax,xstep (on one line):
      0, 20, 1
Enter ymin,ymax,ystep:
      0, 20, 1
Enter equation:
3/2*(cos(3/5*(y-1))+5/4)/(1+(((x-4)/3)2))

```



## 6.3 Fitz

FITZ fits smooth curves to a surface of data points. e.g. Given some data points, e.g.

```

x y z
---- data file testf.dat ----
1 1 1
1 2 1
2 2 1
2 1 1
1.5 1.5 2
-----

```

These points define the four corners of a surface which has one high point in the center. As you can see the order of points is not relevant.

It is important that each line of the data file has three points on it. (x, y, z)

FIT then asks you for a range of x and y values generate z values for. The range is given as minimum,maximum,size.

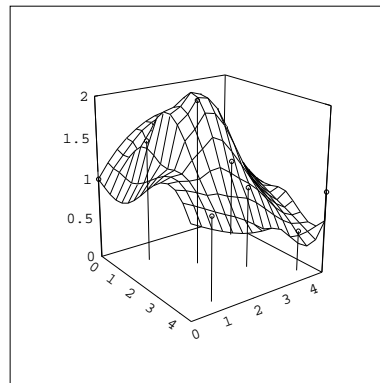
e.g. 1,3,.5 would produce 1, 1.5, 2, 2.5, 3

Use SURF to draw the surface on the screen and see the finished product.

```
$ FITZ
Data file containing x y z data ? TESTF.DAT
Read 5 points from file {testf.dat}
Will write data to {testf.z}
Number of points to use for contouring each point ? [3]
Range of output x values [1,2,6.666667e-02] ? 1,2,.05
Range of output y values [1,2,6.666667e-02] ? 1,2,.05
nx 21, ny 21, Work space iwk=2384 bytes wk=160 bytes
Y = 1 1.05 1.1 1.15 1.2 1.25 1.3 1.35 1.4 1.45 1.5 1.55 1.6 1.65
1.7 1.75 1.8 1.85 1.9 1.95 2
```

Now use TESTF.Z as your data file for SURFACE.

```
begin surface
size 5 5
data fitz1.z
xaxis step 1 hei .2
yaxis step 1 hei .2
zaxis step .5 hei .2
zaxis min 0 max 2
points fitz1/fitz1.dat
droplines lstyle 1
marker circle
view 2.5 3 .3
end surface
```



## 6.4 Contour

This utility produces a graph with contour lines.

For input you give it grid of z values in a file the x and y values are assumed. e.g. a 4x3 grid of points would look like this:

```
! nx 4 ny 3 xmin 0 xmax 0 xmax 6 ymax 0 ymax 10
1 1 1 1
1 2 2 1
1 1 1 1
```

This defines the points

```
0,0,1 2,0,1 4,0,1 6,0,1
0,5,1 2,5,2 4,5,2 6,5,1
0,10,1 2,10,1 4,10,1 6,10,1
```

You would normally generate this file using the utility LETZ or FITZ.

LETZ takes an x,y function and produces a grid of z values

FITZ takes a file of x,y,z data points and fits a grid of z values.

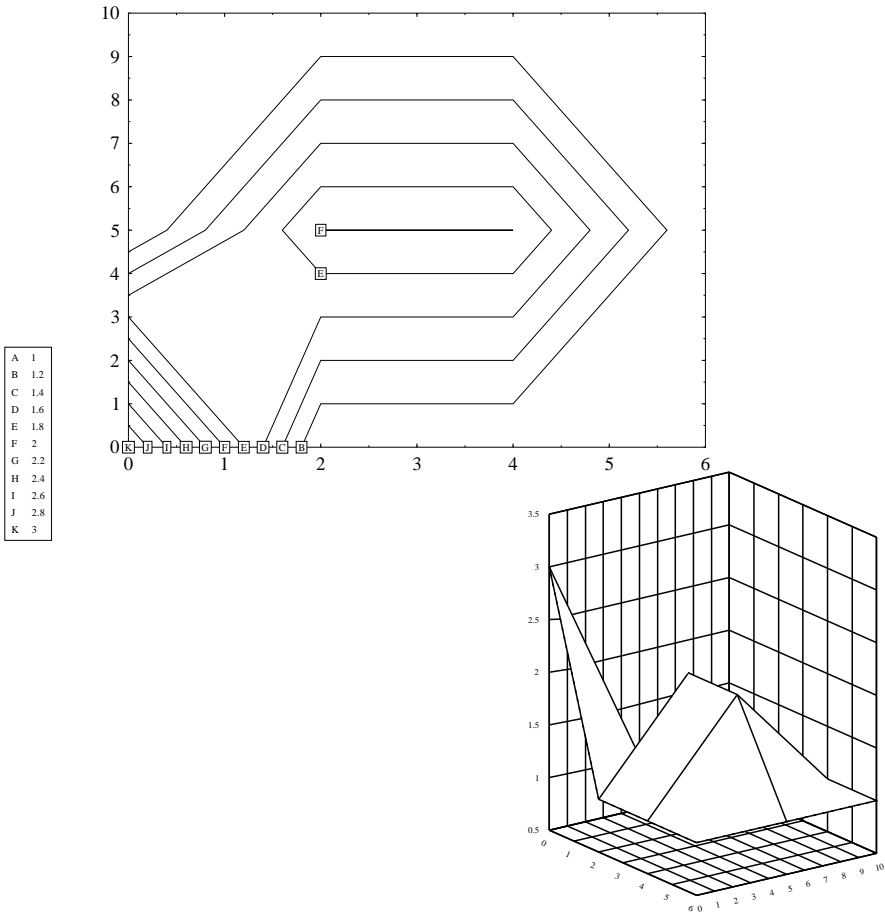
Having created a data file of the correct sort CONTOUR will ask for a list of values to contour at, you can enter a list of values, e.g. "1 2 3 4.5 5" or you can enter a range and step size e.g. "1:3:.5" which would generate "1 1.5 2 2.5 3"

You are then asked if you want the lines labeled with the letters A ... Z or by the actual numbers.

Then you are asked for a file name to write output to. If you give the name XYZ then CONTOUR will create:

xyz.gle	The gle file to draw the contour plot
xyz.dat	The data file containing the contour lines
xyz.lab	A gle include file which labels the graph
xyz.key	A gle include file which draws a key

Then wait while the contours are calculated, this takes some time (even on a fast PC) e.g. 10 seconds on 33Mhz 386. The exact time will depend on the number of points and number of contour lines you are using.



## Chapter 7

# GLE Utilities

### 7.1 Fitls

The FITLS utility allows an equation with  $n$  unknown constants to be fitted to experimental data.

For example to fit a simple least squares regression line to a set of points you would give FITLS the equation:  $a*x+b$

FITLS would then solve the equation to find the *best* values for the constants  $a$  and  $b$ .

FITLS can work with non linear equations, it will ask for initial values for the parameters so that a solution around those initial guesses will be found.

FITLS writes out a GLE file containing commands to draw the data points and the equation it has fitted to them.

Here is a sample FITLS session:

```
$ fitls
Input data file (x and y columns optional) [test.dat,1,2] ? fitls.dat
Loading data from file, fitls.dat, xcolumn=1, ycolumn=2
Valid operators: +, -, *, /, ^ (power)
Valid functions:
    abs(), atn(), cos(), exp(), fix(), int()
    log(), log10(), not(), rnd(), sgn(), sin()
    sqr(), sqrt(), tan()

Enter a function of 'x' using constants 'a'...'z'
e.g.    a + b*x    (standard linear least squares fit)
        sin(x)*a+b
```

```
a + b*x + c*x^2 + d*x^3
log(a*x)+(b+x)*c+a
```

Equation ?  $\sin(a*x)*b+c*x^2+d$

Output file name to write gle file [fitls.gle] ?

Precision of fit required, [1e-4] ?

Initial value for constant a [1.0] ?

Initial value for constant b [1.0] ?

Initial value for constant c [1.0] ?

Initial value for constant d [1.0] ?

0 evaluations, 1 1 1 1 , fit = 1355.36

20 evaluations, 1.97005 1 1 1 , fit = 1281.95

40 evaluations, 1.97005 10.228 0.151285 1 , fit = 54.7694

60 evaluations, 2.01053 10.228 0.151285 1.06365 , fit = 54.1771

.

.

.

440 evaluations, -0.640525 -2.81525 0.13997 1.13871 , fit = 0.940192

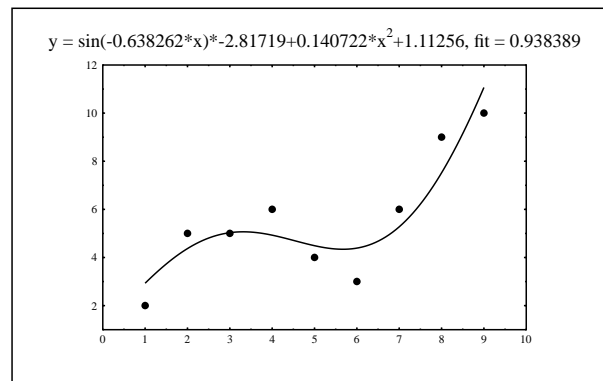
460 evaluations, -0.638055 -2.82934 0.140971 1.10502 , fit = 0.93842

480 evaluations, -0.63808 -2.82357 0.140993 1.10452 , fit = 0.938389

a = -0.638262 b = -2.81719 c = 0.140722 d = 1.11256

10 Iterations, sum of squares divided by n = 0.938389

$y = \sin(-0.638262*x)*-2.81719+0.140722*x^2+1.11256$



## 7.2 Manip

Manip is a data manipulation package. It reads in a text file of numbers and displays them like a spreadsheet. You can then do simple operations on the columns and write them out in any format you like.

### 7.2.1 Usage

```
MANIP infile.dat -recover -step -commands c.log -single -size x y
```

#### -recover

Manip logs everything you type to a file called `MANIP_.J1`. When you use the `-RECOVER` option on the `manip` command it then reads keys from that file as if they were typed at the keyboard.

This will restore you to the point just before your pc crashed. The last three journal files are stored (`.j1 .j2 .j3`) simply copy the one you want to (`.j1`) to use it.

#### -step

Used with `recover`, press a space for each key you want to read from the journal file, press any other key to stop reading the journal.

#### -commands *filename.man*

This reads the commands in *filename.man* as if they were typed at the keyboard.

#### -single

This makes `MANIP` use single precision arithmetic and doesn't store strings at all, this enables three times as much data in the same amount of memory.

#### -size *x y*

Sets the initial size of the spreadsheet. Use this with large datasets as it prevents the heap from becoming fragmented and thus lets you use much larger datasets.

## Range

Most `manip` commands accept a range as one or more of their parameters. A range is a rectangular section of your spreadsheet. A range can either start with a 'c' or an 'r' and this will affect how the command operates.

If your spreadsheet has 5 columns and 10 rows then.

```
c1 == c1c1r1r10 == 1,1 1,2 1,3 1,4 1,5 1,6 ...
r1 == r1r1c1c5  == 1,1 2,1 3,1 4,1 5,1
c1c2 == c1c2r1r10 == 1,1 2,1 1,2 2,2 3,1 3,2 ...
r1r2c3 == r1r2c3c5 == 3,1 3,2 4,1 4,2 5,1 5,2
```

## Arrows

The arrow keys normally move the data cursor, however if you are half way thru typing a command then, the left and right arrow keys allow you to edit the command. Use the `PAGE-UP` and `PAGE-DOWN` keys to recall your last command.

`SHIFT` arrow keys will jump 7 cells at a time for fast movement.

Further help is available on the following topics via the HELP command e.g. "HELP COPY"

### 7.2.2 Manip Primitives (a summary)

```
@mycmds
Arrows
BLANK
CLEAR
CLOSE
COPY [range] [range] IF [exp]
DATA [range]
DELETE [range] IF [exp]
EXIT filename [range] -TAB -SPACE -COMMA
FIT c3
Functions
GENERATE [pattern] [destination]
GOTO x y
INSERT [Cn] or [Rn]
LOAD filename [range] -0
LOAD filename [range]
LOGGING mycmds.man
MOVE [range] [range] IF [exp]
NEW
PARSUM [range1] [range2]
PROP [range] [range]
QUIT
Recover (recovering from power failure or crash)
SAVE filename [range] -TAB -SPACE -COMMA
SET SIZE ncols nrow
SET BETWEEN " "
SET COLTYPE
SET COLWIDTH
SET NCOL n
SET DPOINTS n
SET DIGITS n
SET WIDTH n
SHELL
SORT [range] on [exp]
SUM [range]
SWAP CnCn — RnRn
```

### 7.2.3 Manip Primitives (in detail)

COPY [range1] [range2] if [exp]

For copying a section to another section. They do not have to be the same shape. The pointers to both ranges are increased even if the number is not copied e.g.

```
"% COPY r4r2 r1r2"
"% COPY c1c3r6r100 c6c8 if c1<c2"
```

```
"% COPY C1 C2 IF C1<4"
c1  c2
1      1
2      2
5      -
3      3
9      -
```

### DELETE [*range*] IF [*exp*]

For deleting entire rows or columns. e.g.

```
"% DELETE c1c3 IF r1>3.and.r2=0
"% DELETE r1"
```

Numbers are shuffled in from the right to take the place of the deleted range.

### DATA [*range*]

Data entry mode is useful for entering data. After typing in "% DATA c1c3" or "% DATA C2" you can then enter data and pressing `↓` will move you to the next valid data position. In this mode text or numbers can be entered. Press ESC to get back to command mode.

### FIT *c3*

"FIT C3" will fit a least squares regression line to the data in columns c3 and c4 (x values taken from c3) and print out the results.

### EXIT

EXIT saves the data in your input file spec and exits to DOS. You can optionally specify an output file as well. eg. "% EXIT myfile.dat"

The command "EXIT myfile.dat c3c5r1r3" will write out that range of numbers to the file.

By default manip will write columns separated by spaces.

The command "EXIT myfile.dat -TAB" will put a single tab between each column of numbers and "EXIT myfile.dat -COMMA" will put a comma and a space between each number. (these two options are useful if your data file is very big and you don't want to waste disk space with the space characters.) Note: The settings stay in effect for future saves and exits.

You can make it line up the columns on the decimal point by typing in the command. "SET DPOINTS 3"

You change the width of each column or completely remove the spaces between columns with the command. "SET WIDTH 10" (or set width 0)

You can change the number of significant digits displayed with the command "SET DIGITS 4"

**SAVE** *myfile.dat*

Saves all or part of your data. The command "SAVE myfile.dat c3c5r1r3" will write out that range of numbers to the file.

By default manip will write columns separated by spaces.

The command "SAVE myfile.dat -TAB" will put a single tab between each column of numbers and "SAVE myfile.dat -COMMA" will put a comma and a space between each number. (these two options are useful if your data file is very big and you don't want to waste disk space with the space characters).

Further options are the same like EXIT

**GOTO**

For moving the cursor directly to a point in your array. e.g. "% GOTO x y"

**CLEAR**

"% CLEAR C2C3" Clears the given range of all values

**BLANK**

"% BLANK C2C3" Clears the given range of all values

**NEW**

Clears the spread sheet of all data and frees memory.

**INSERT**

Inserts a new column or row and shifts all others over.  
e.g. "% INSERT c5" or "% INSERT r2".

**LOAD**

Load data into columns. eg. "% LOAD filename" loads all data into corresponding columns. "% LOAD filename c3" load first column of data into c3 etc.

"LOAD myfile.dat c3 -LIST" This command will load the data into a single column or range (even if it is several columns wide in the data file)

**MOVE** [*range1*] [*range2*] if [*exp*]

For copying a section to another section. They do not have to be the same shape. The pointer to the destination is only increased if the line or column is copied e.g.

```
"% MOVE c1 c2c3"
"% MOVE r4r2 r1r2"
"% MOVE c1c3r6r100 c6c8 if c1<c2"
```

```
"% MOVE C1 C2 IF C1<4"
```

```
c1 c2
```

```
1 1
```

```
2 2
```

```
5 3
```

```
3 -
```

```
9 -
```

(See COPY command)

**SORT** [*range*] ON [*exp*]

Sort entire rows of the data based on the data in a particular column. e.g.

```
"% SORT c8 on c9"
"% SORT c1c8 on -c8"
"% SORT c1c3 on c2 " !for sorting strings
```

This command works out how to sort the column (or exp) specified in the ON part of the command. It then does that operation to the range specified. e.g. "SORT C1 ON C1" will sort column one.

Use the additional qualifier **-STRINGS** if you want to sort a column with strings in it. e.g. "sort c1 on c2 -strings"

**SWAP**

Swap over two columns or rows. e.g.

```
"% SWAP c1c2"
"% SWAP r3r1"
```

**SET SIZE** *ncols nrows*

"SIZE 3 4" Truncates the spreadsheet to 3 columns and 4 rows. This also sets the values to use for default ranges.

**SET BETWEEN** " "

"SET BETWEEN "##" Defines the string to be printed between each column of numbers when written to a file. This is normally set to a single space.

**SET COLWIDTH**

Set the width of each column when displayed. e.g. "% SET COLWIDTH 12"

**SET COLTYPE** [*n*] DECIMAL — EXP — BOTH — DPOINTS *n*

This commands allows all or individual columns to be set to different output types. If colnumber is missing then that setting is applied to all columns.

SET COLTYPE Ccolnumber TYPE Where TYPE is one of:

```
DECIMAL    produces 123.456
EXP        produces 1.23456e02
BOTH       produces whichever is more suitable
DPOINTS n  produces a fixed number of decimal places.
e.g.
SET COLTYPE c2 DECIMAL
SET COLTYPE c1 EXP
SET COLTYPE c3 DPOINTS 4
```

Would print out: 1.2e02    1.2    1.2000

SET COLTYPE EXP    (column number missed out)

Would print out: 1.2e02    1.2e02    1.2e02

**SET NCOL *n***

Set the number of columns to display. e.g. "% SET NCOL 3"

**SET DPOINTS *n***

Sets the number of decimal places to print. This is used for producing columns which line up on the decimal point. e.g. with DPOINTS 3.

```
2.2    -> 2.200
234    -> 234.000
```

(See also SET COLTYPE)

**SET DIGITS *n***

Sets the number of significant digits to be displayed, e.g. with DIGITS 3.

```
123456    becomes    123000
0.12345    becomes    0.123
```

**SET WIDTH *n***

Sets the width of padding to use for the columns when they are written to a file. The columns usually one space wider than this setting as the BETWEEN string is usually set to one space by default.

**LOGGING**

For creating command files. e.g.

```
"% LOG sin.man"
"% c2=sin(c1)
"% c3=c2+2
"% close"
```

Then type in "@sin" to execute these commands.

**PROPAGATE [*source*] [*destination*]**

This command has the same format as move. The difference is that the source is copied as many times as possible to fill up the destination. e.g. "% PROP c1r1r7 c2"

**SUM [*range*]**

Adds up all the numbers in a range and displays the total and average. e.g. "% SUM C1C3"

**PARSUM [*range1*] [*range2*]**

Adds up one column, putting the partial sum's into another column. e.g. 1,2,3,4 becomes 1,3,6,10.

**GENERATE [*pattern*] [*destination*]**

For generating a pattern of data e.g. 1 1 2 2 5 5 1 1 2 2 5 5 etc.

```
"% GEN 2(1,2,5)30 c4" !1 1 2 2 5 5 repeated 30 times
"% GEN (1:100:5)5 c1" !1 to 100 step 5, 5 times
"% GEN (1,2,*,3:5)5 c1" !missing values included
```

### Functions

Calculations can be performed on rows or columns. eg "% C1=C2\*3+R" where "R" stands for row-number and C1 and C2 are columns. They can also be performed on ROWS. eg

```
r1=sin(r2)+log10(c)
c1 = cell(c+1,r)+cell(c+2,r)
cell(1,3) = 33.3
3+4*COS(PI/180)^(3+1/30)+C1+R
```

Valid operators and functions:

,	+	-	^	*	/	<=
>=	<>	<	>	=	)AND(	)OR(
ABS(	ATN(	COS(	EXP(	FIX(	INT(	LOG(
LOG10(	SGN(	SIN(	SQR(	TAN(	NOT(	RND(
SQRT(	.NE.	.EQ.	.LT.	.GT.	.LE.	.GE.
.NOT.	.AND.	.OR.				

### QUIT

Abandon file.

### SHELL




























Gives access to DOS.



# Appendix A

## Tables

### A.1 Markers

	—	circle		—	dag
	—	triangle		—	ddag
	—	square		—	asterisk
	—	diamond		—	oplus
	—	fcircle		—	ominus
	—	ftriangle		—	otimes
	—	fsquare		—	odot
	—	fdiamond		—	trianglez
	—	dot		—	diamondz
	—	cross		—	wcircle
	—	club		—	wtriangle
	—	heart		—	wsquare
	—	star		—	wdiamond
	—	snake			

### A.2 Fonts

rm	Roman	psagb	AvantGarde-Book
rmb	<b>Roman Bold</b>	psagbo	<i>AvantGarde-BookOblique</i>
rmi	<i>Roman Italic</i>	psagd	<b>AvantGarde-Demi</b>
ss	Sans Serif	psagdo	<b><i>AvantGarde-DemiOblique</i></b>
ssb	<b>Sans Serif Bold</b>	psbd	<b>Bookman-Demi</b>
ssi	<i>Sans Serif Italic</i>	psbdi	<b><i>Bookman-DemiItalic</i></b>
tt	Typewriter	psbli	<i>Bookman-LightItalic</i>
ttb	<b>Typewriter Bold</b>	psc	Courier
tti	<i>Typewriter Italic</i>	pscb	<b>Courier-Bold</b>
		pscbo	<b><i>Courier-BoldOblique</i></b>
		psco	<i>Courier-Oblique</i>
texcmb	<b>Computer Modern Bold</b>	psh	Helvetica
texcmitt	<i>Computer Modern Italic Typewriter</i>	pshb	<b>Helvetica-Bold</b>
texcmmi	<i>ComputerModernMathsItalic</i>	pshbo	<b><i>Helvetica-BoldOblique</i></b>
texcmr	Computer Modern Roman	psho	<i>Helvetica-Oblique</i>
texcmss	Computer Modern Sans Serif	pshc	Helvetica-Condensed
texcmssb	<b>Computer Modern Sans Serif Bold</b>	pshcb	<b>Helvetica-Condensed-Bold</b>
texcmssi	<i>Computer Modern Sans Serif Italic</i>	pshcdo	<i>Helvetica-Condensed-BoldOblique</i>
texcmti	<i>Computer Modern Text Italic</i>	pshn	Helvetica-Narrow
texcmtt	Computer Modern Typewriter Text	pshnb	<b>Helvetica-Narrow-Bold</b>
		pshnbo	<b><i>Helvetica-Narrow-BoldOblique</i></b>
		pshno	<i>Helvetica-NarrowOblique</i>
plba	<b>Block Ascii</b>	psncsb	<b>NewCenturySchlbk-Bold</b>
plci	<i>Complex Italic</i>	psncsbi	<b><i>NewCenturySchlbk-BoldItalic</i></b>
plcr	Complex Roman	psncsi	<i>NewCenturySchlbk-Italic</i>
plcs	<i>Complex Script</i>	psncsr	NewCenturySchlbk-Roman
pldr	Duplex Roman	pspb	<b>Palatino-Bold</b>
plge	<b>Gothic English</b>	pspbi	<b><i>Palatino-BoldItalic</i></b>
plgg	<b>Gothic German</b>	pspi	<i>Palatino-Italic</i>
plgi	<b>Gothic Italian</b>	pspr	Palatino-Roman
plsa	Simplex Ascii	pssym	Συμβολ
plsg	Τινπμεω Ηεσναξ	pstb	<b>Times-Bold</b>
plsr	Simplex Roman	pstbi	<b><i>Times-BoldItalic</i></b>
plss	<i>Simplex Script</i>	psti	<i>Times-Italic</i>
plti	<i>Triplex Italic</i>	pstr	Times-Roman
pltr	<b>Triplex Roman</b>	pszcmi	<i>ZapfChancery-MediumItalic</i>

## A.3 Functions

Function Name	Returns ...
<code>abs(exp)</code>	absolute value of expression
<code>acos(exp)</code>	arccosine
<code>acosh(exp)</code>	inverse hyperbolic cosine
<code>acot(exp)</code>	$1/\text{atan}(\text{exp})$
<code>acoth(exp)</code>	$1/\text{atanh}(\text{exp})$
<code>acsc(exp)</code>	$1/\text{asin}(\text{exp})$
<code>acsch(exp)</code>	$1/\text{asinh}(\text{exp})$
<code>asec(exp)</code>	$1/\text{acos}(\text{exp})$
<code>asech(exp)</code>	$1/\text{acosh}(\text{exp})$
<code>asin(exp)</code>	arcsine
<code>asinh(exp)</code>	inverse hyperbolic sine
<code>atan(exp)</code>	arctan
<code>atanh(exp)</code>	inverse hyperbolic tangent
<code>atn(exp)</code>	same as <code>ATAN(exp)</code>
<code>cos(exp)</code>	cosine
<code>cosh(exp)</code>	hyperbolic cosine
<code>cot(exp)</code>	$1/\text{tan}(\text{exp})$
<code>coth(exp)</code>	$1/\text{tanh}(\text{exp})$
<code>csc()</code>	$1/\text{sin}(\text{exp})$
<code>csch(exp)</code>	$1/\text{sinh}(\text{exp})$
<code>cvtrgb(red,green,blue)</code>	create color given RGB values
<code>date\$()</code>	current date e.g. "Tue Apr 09 1991"
<code>device\$()</code>	available devices e.g. "HARDCOPY, PS"
<code>exp(exp)</code>	exponent
<code>fix(exp)</code>	<code>exp</code> rounded towards 0
<code>format\$(exp,format)</code>	format <code>exp</code> as specified in <code>format</code> (page 21)
<code>height(name\$)</code>	the height of the object <code>name\$</code>
<code>int(exp)</code>	integer part of <code>exp</code>
<code>left\$(str\$,exp)</code>	left <code>exp</code> characters of <code>str\$</code>
<code>len(str\$)</code>	the length of <code>str\$</code>
<code>log(exp)</code>	log to base $e$ of <code>exp</code>
<code>log10(exp)</code>	log to base 10 of <code>exp</code>
<code>not(exp)</code>	logical not of <code>exp</code>
<code>num1\$(exp)</code>	as above but with no spaces

<code>num\$(exp)</code>	string representation of <code>exp</code>
<code>pageheight()</code>	the height of the page (from <code>size</code> command)
<code>pagewidth()</code>	the width of the page (from <code>size</code> command)
<code>pointx(pt)</code>	the x value of point <code>pt</code>
<code>pointy(pt)</code>	the y value of point <code>pt</code>
<code>pos(str1\$,str2\$,exp)</code>	position of <code>str2\$</code> in <code>str1\$</code> from <code>exp</code>
<code>right\$(str\$,exp)</code>	rest of <code>str\$</code> starting at <code>exp</code>
<code>rnd(exp)</code>	random number from seed <code>exp</code>
<code>sec(exp)</code>	$1/\cos(\exp)$
<code>sech(exp)</code>	$1/\cosh(\exp)$
<code>seg\$(str\$,exp1,exp2)</code>	<code>str\$</code> from <code>exp1</code> to <code>exp2</code>
<code>sgn(exp)</code>	returns 1 if <code>exp</code> is positive, -1 if <code>exp</code> is negative
<code>sin(exp)</code>	sine of <code>exp</code>
<code>sinh(exp)</code>	hyperbolic sine
<code>sqr(exp)</code>	<code>exp</code> squared
<code>sqrt(exp)</code>	square root of <code>exp</code>
<code>tan(exp)</code>	tangent of <code>exp</code>
<code>tanh(exp)</code>	hyperbolic tangent
<code>tdepth(str\$)</code>	the depth of <code>str\$</code> assuming current the font, size
<code>theight(str\$)</code>	the height of <code>str\$</code> assuming current font, size
<code>time\$()</code>	current time e.g. "11:44:27"
<code>todeg(exp)</code>	convert from radians to degrees
<code>torad(exp)</code>	convert from degrees to radians
<code>twidth(str\$)</code>	the width of <code>str\$</code> assuming current font, size
<code>val(str\$)</code>	value of the string <code>str\$</code>
<code>width(name\$)</code>	the width of the object <code>name\$</code>
<code>xend()</code>	the x end point of a text string when drawn
<code>xg(xexp)</code>	converts units of last graph to abs cm.
<code>xpos()</code>	the current x point
<code>yend()</code>	the y end point of a text string when drawn
<code>yg(yexp)</code>	converts units of last graph to abs cm.
<code>ypos()</code>	the current y point

## A.4 L<sup>A</sup>T<sub>E</sub>X Macros and Symbols

There are several L<sup>A</sup>T<sub>E</sub>X like commands which can be used within text, they are:

<code>\ ' \v \u \= \^</code>	Implemented TeX accents
<code>\. \H \~ \"</code>	
<code>^{} </code>	Superscript
<code>_{} </code>	Subscript
<code>\\</code>	Forced Newline
<code>\_</code>	Underscore character
<code>\,</code>	.5em (em = width of the letter 'm')
<code>\:</code>	1em space
<code>\;</code>	2em space
<code>\tex{expression}</code>	Any LaTeX expression
<code>\char{22}</code>	Any character in current font
<code>\chardef{a}{hello}</code>	Define a character as a macro
<code>\def\v{hello}</code>	Defines a macro
<code>\movexy{2}{3}</code>	Moves the current text point
<code>\glass</code>	Makes move/space work on beginning of line
<code>\rule{2}{4}</code>	Draws a filled in box, 2cm by 4cm
<code>\setfont{rmb}</code>	Sets the current text font
<code>\sethei{.3}</code>	Sets the font height (in cm)
<code>\setstretch{2}</code>	Scales the quantity of glue between words
<code>\lineskip{.1}</code>	Sets the default distance between lines of text
<code>\linegap{-1}</code>	Sets the minimum required gap between lines

[	\lbrack	ß	\ss	≥	\geq	†	\dag
‡	\ddag	§	\S	¶	\P	©	\copyright
α	\alpha	β	\beta	γ	\gamma	δ	\delta
ε	\epsilon	ζ	\zeta	η	\eta	θ	\theta
ι	\iota	κ	\kappa	λ	\lambda	μ	\mu
ν	\nu	ξ	\xi	π	\pi	ρ	\rho
σ	\sigma	τ	\tau	υ	\upsilon	φ	\phi
χ	\chi	ψ	\psi	ω	\omega	ε	\varepsilon
θ	\vartheta	ϖ	\varpi	ρ	\varrho	ς	\varsigma
φ	\varphi						

ℵ	\aleph	ι	\imath	ℐ	\jmath
ℓ	\ell	℘	\wp	ℜ	\Re
ℑ	\Im	∂	\partial	∞	\infty
′	\prime	∅	\emptyset	∇	\nabla
⊤	\top	⊥	\bot	△	\triangle
∀	\forall	∃	\exists	¬	\neg
♭	\flat	‡	\natural	‡	\sharp
♣	\clubsuit	◇	\diamondsuit	♥	\heartsuit
♠	\spadesuit	∏	\coprod	∨	\bigvee
∧	\bigwedge	⊕	\biguplus	∩	\bigcap
∪	\bigcup	∏	\prod	∑	\sum
⊗	\bigotimes	⊕	\bigoplus	⊙	\bigodot
⊔	\bigsqcup	∫	\smallint	∫	\intop
∫	\oint	◁	\triangleleft	▷	\triangleright
△	\bigtriangleup	▽	\bigtriangledown	∧	\wedge
∧	\land	∨	\vee	∨	\lor
∩	\cap	∪	\cup	‡	\ddagger
†	\dagger	⊔	\sqcup	⊔	\sqcup
⊕	\uplus	∏	\amalg	◇	\diamond
•	\bullet	ℓ	\wr	÷	\div
⊙	\odot	⊗	\oslash	⊗	\otimes
⊖	\ominus	⊕	\oplus	≠	\mp
±	\pm	○	\circ	○	\bigcirc
\	\setminus	·	\cdot	*	\ast
×	\times	*	\star		

## A.5 Fonttables

```

!From: TREV::SRPDBRS      27-JAN-1992 16:30:58.78
!To: GRV::SRGHCXP
!CC:
!Subj: Gle font viewer - here's a going one
!Chris,
! thanks for the first approximation of the 'see a font' program
!   - this is a working (ie tested and going) version. It could be
!   useful.
! Bruce
!PS - pszd isn't available on (my) pc so it didn't work there !!
!-----
size 18 24
text \chardef~{\movexy{.4}{0}}
sub tt a$
  write a$
  rmove 0 -1.2
end sub

set font pszd
amove .5 23.5
for j = 0 to 15
  for i = 0 to 15
! note: no spaces in expression below. (except inside quotes)
    xx$ = "\char{" + num1$(j*16+i) + "}"
    write xx$
    rmove .9 0
  next i
  amove .5 (23.5 - (1.3*(j+1)))
next j

amove 14 .5
set hei .2 font ss
write "DING2.GLE - BRS "+DATE$()

```

showfont.gle






























# UGE Wall Reference

_____ 0	_____ 5
_____ 1	_____ 6
_____ 2	_____ 7
_____ 3	_____ 8
_____ 4	_____ 9

\_\_\_\_\_ 9229

lwidth 0.2  
 lwidth 0.1  
 lwidth 0.05  
 lwidth 0.02  
 lwidth 0.01  
 lwidth 0.0001  
 lwidth 0

[illegible]

	circle		dag
	triangle		ddag
	square		asterisk
	diamond		oplus
	fcircle		ominus
	ftriangle		otimes
	fsquare		odot
	fdiamond		trianglez
	dot		diamondz
	cross		wcircle
	club		wtriangle
	heart		wsquare
	star		wdiamond
	snake		

The GRID and SHADE patterns should only be used for filling on PostScript printers, the grey levels and colors will work for both filling and color settings on any device.

	Grid5		Grey90
	Grid4		Grey20
	Grid3		Grey10
	Grid2		Grey5
	Grid1		Grey1
	Grid		White
	Shade5		Black
	Shade4		Yellow
	Shade3		Magenta
	Shade2		Blue
	Shade1		Green
	Shade		Red

