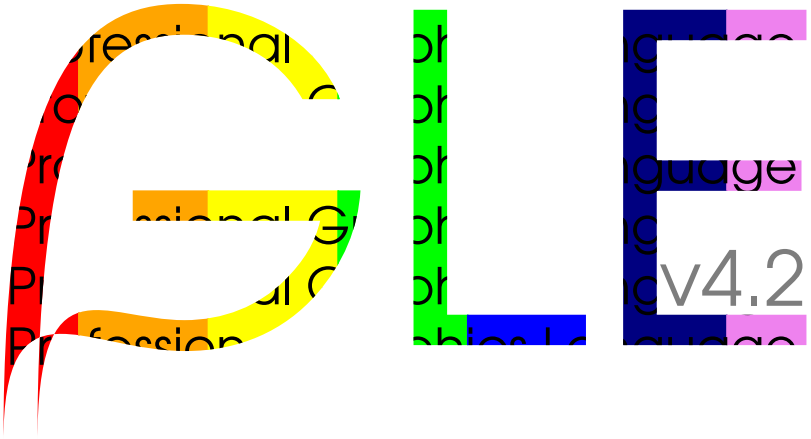


Graphics Layout Engine

GLE 4.2.4c

User Manual

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

http://www.gle-graphics.org/

August 12, 2013

Contents

Chapter 1

Preface

Abstract

GLE (Graphics Layout Engine) is a graphics scripting language designed for creating publication quality graphs, plots, diagrams, figures and slides. GLE supports various graph types (function plots, histograms, bar graphs, scatter plots, contour lines, color maps, surface plots, ...) through a simple but flexible set of graphing commands. More complex output can be created by relying on GLE's scripting language, which is full featured with subroutines, variables, and logic control. GLE relies on LaTeX for text output and supports mathematical formulea in graphs and figures. GLE's output formats include EPS, PS, PDF, JPEG, and PNG. GLE is licenced under the BSD license. QGLE, the GLE user interface, is licenced under the GPL license.

Trademark Acknowledgements

The following trademarks are used in this manual.

Windows	Microsoft Corporation.
TeX	Donald E. Knuth, A Typesetting System.
LaTeX	Leslie Lamport, A Document Preparation System.
PostScript	Page Description Language, Adobe Systems Inc.

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.
keyword	Keywords are represented in a bold typewriter font.
exp,x,y,x1,y1	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 2, The GLE Tutorial (beginners only).
2. Examine the examples at `http://www.gle-graphics.org/examples/`.
3. Browse through Chapter 4, The Graph Module.

For those with time:

- **Chapter 2, GLE Tutorial:**
  Covers installation and drawing a simple graph, highly recommended if you have never used GLE before.
- **Chapter 3, GLE Primitives:**
  Describes the commands used for creating diagrams and slides and for annotating graphs.

- **Chapter 4, The Graph Module:**
  Describes the commands for drawing graphs.
- **Chapter 5, The Key Module:**
  Describes the commands for producing keys for graphs.
- **Chapter 7, Advanced Features of GLE:**
  Covers advanced features of GLE. This includes programming constructs, the LaTeX interface, ...
- **Chapter 9, Surface and Contour Plots:**
  Describes the commands for drawing three-dimensional graphs.
- **Chapter 10, GLE Utilities:**
  Describes FITLS and MANIP.

Chapter 2

Tutorial

2.1 Installing GLE

This tutorial assumes that GLE is correctly installed. Information about how to install GLE can be found at the following URLs. The GLE distribution also includes a README with brief installation instructions.

- Installation on Windows: `http://www.gle-graphics.org/tut/windows.html`.
- Installation on Linux: `http://www.gle-graphics.org/tut/linux.html`.
- Installation on Mac OS/X: `http://www.gle-graphics.org/tut/mac.html`.

Feel free to post any questions or comments you might have about installing GLE on the GLE mailing list, which is available here:

- Mailing list: `https://lists.sourceforge.net/lists/listinfo/glx-general`.

2.2 Running GLE

GLE is essentially a command line application; this tutorial will show you how to run it from the command prompt. GLE can also be run from your favorite text editor or from QGLE, GLE's graphical user interface. More information about running GLE from a text editor is given in the installation instructions.
On Windows, you run GLE from the Windows Command Prompt. Normally the GLE installer should have added an entry labeled "Command Prompt" to GLE's folder in the start menu. On Unix-like operating systems, GLE runs from an X-terminal, such as "konsole" on Linux / KDE.
Once you have opened the command prompt or terminal, try running GLE by entering the following command.

```
gle
```

As a result, GLE displays the following message.

```
GLE version x.y.z
Usage: gle [options] filename.gle
More information: gle -help
```

If this message does not appear and you see an error message instead, then GLE is not correctly installed. Refer to the installation instructions (Appendix A.5) for more information. In the following, we will show how to construct a simple drawing with GLE.

2.3 Drawing a Line on a Page

Let's start with drawing a line on the page. GLE needs to know the size of the drawing you wish to make. This is accomplished with the size command:

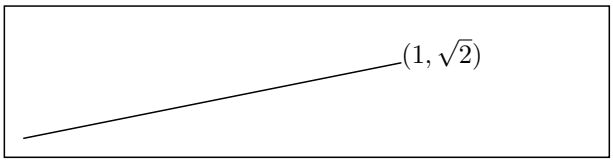


Figure 2.1: Result of your first GLE script.

```
size  8 2
```

This specifies that the output will be 8cm wide and 2cm high. Next we define a "current point" by moving to somewhere on the page:

```
amove 0.25 0.25
```

The origin (0,0) is at the bottom left hand corner of the page. Suppose we wish to draw a line from this point 5 cm across and 1 cm up:

```
size  8 2
amove 0.25 0.25
rline 5 1
```

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

```
size  8 2
amove 0.25 0.25
aline 5.25 1.25
```

To draw some text on the page at the current point, use the write command:

```
 write "Hi there"
```

Or, alternatively, you could include arbitrary LaTeX expressions using the tex command:

```
tex "$(1,\sqrt{2})$"
```

Now we have constructed complete GLE script, which looks as follows:

```
size 8 2 box
amove 0.25 0.25
rline 5 1
tex "$(1,\sqrt{2})$"
```

Enter the above GLE script using a text editor and save it to disk (any editor that saves in UTF8 or ASCII format will work). The following assumes that you have saved the file under the name "`test.gle`" in the folder `C:\GLE` on Windows, or `/home/john/gle` on a Unix-like operating system. Now open a command prompt and go to the folder where you saved the file. Then, run GLE on the file.
On Windows, you do this as follows (`C:\>` is the prompt):

```
C:\> cd C:\GLE
C:\GLE> gle test.gle
```

Or on Unix:

```
cd ~/gle
gle test.gle
```

GLE produces by default an Encapsulated PostScript (.eps) file:

```
 GLE x.y.z [test.gle]-C-R-[test.eps]
```

Try viewing the resulting "`test.eps`" with a PostScript viewer such as GhostView, and compare it to the output shown in Fig. 2.1. You can also preview it with QGLE, GLE's graphical user interface. After you've started QGLE, enter the following command at the command prompt.

```
gle -p test.gle
```

This will preview the output in the QGLE previewer window. GLE can also create PDF files. This is accomplished by setting the output device to "pdf".

```
gle -device pdf test.gle
```

Try viewing the resulting "`test.pdf`" with Acrobat Reader or similar. Other output formats supported by GLE (eps, ps, pdf, svg, jpg, png, x11) can also 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:

```
gle -d jpg -r 200 test.gle
```

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

```
gle -help
```

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

```
gle -help option
```

The following command line options are supported by GLE:

```
-help           Shows help about command line options
-info           Outputs software version, build date, GLE\_TOP, GLE\_BIN, etc.
-verbosity      Sets the verbosity level of GLE console output
-device         Selects output device(s)
-cairo          Use cairo output device
-resolution     Sets the resolution for bitmap and PDF output
-fullpage       Selects full page output
-landscape      Selects full page landscape output
-output         Specifies the name of the output file
-nosave         Don't write output file to disk (dry-run)
-preview        Previews the output with QGLE
-gs             Previews the output with GhostScript
-version        Selects a GLE version to run
-compatibility  Selects a GLE compatibility mode
-calc           Runs GLE in "calculator" mode
-catcsv         Pretty print a CSV file to standard output
-tex            Indicates that the script includes LaTeX expressions
-inc            Creates an .inc file with LaTeX code
-texincprefix   Adds the given subdirectory to the path in the .inc file
-mkinittex      Creates "inittex.ini" from "init.tex"
-finddeps       Automatically finds dependencies
-nocolor        Forces grayscale output
-transparent    Creates transparent output (with -d png)
-noctrl-d       Excludes CTRL-D from the PostScript output
-nomaxpath      Disables the upper-bound on the drawing path complexity
-noligatures    Disable the use of ligatures for 'fl' and 'fi'
-gsoptions      Specify additional options for GhostScript
-safemode       Disables reading/writing to the file system
-allowread      Allows reading from the given path
-allowwrite     Allows writing to the given path
-keep           Don't delete temporary files
```

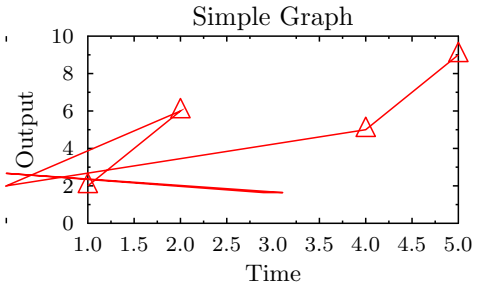
2.4 Drawing a Simple Graph

This section shows how to go about drawing a simple graph. Enter the following data in a new file and save it as "test.csv". Note that you can export files in CSV (comma separated values) format with most spread sheet programs.

```
1,2
2,6
3,2
4,5
5,9
```

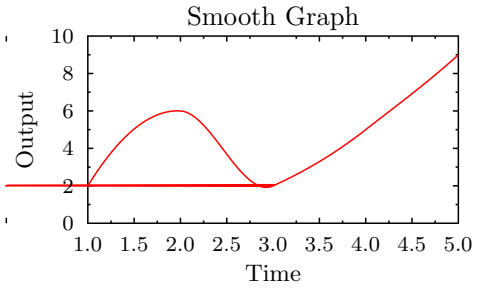
The data is in two columns with a comma separating each column of numbers. The following commands will draw a simple line graph of the data.

```
size 7 4
begin graph
   title  "Simple Graph"
   xtitle "Time"
   ytitle "Output"
   data   "test.csv"
   d1 line marker triangle color red
end graph
```



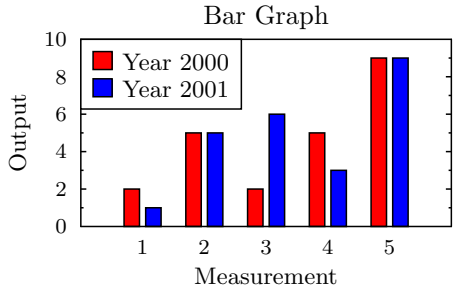
The commands title, xtitle, and ytitle specify the graph title and the axis titles. The command data loads the data file and the d1 command specifies how the first curve on the graph should look like. These commands are discussed in detail in Chapter 4. Possible values for the marker option can be found on the GLE wall reference chart in Appendix A.9.
The axis ranges can be specified with "xaxis min v_0 max v_1" and "yaxis min v_0 max v_1". A smooth line can be drawn through the data points by changing the d1 command to: "d1 line smooth" as in the following example.

```
size 7 4
begin graph
   title  "Smooth Graph"
   xtitle "Time"
   ytitle "Output"
   data   "test.csv"
   yaxis min 0 max 10
   d1 line smooth color red
end graph
```



Note that the order of the commands is not important, except that circle is a parameter for the option marker and therefore must come right after it. The same holds for line and smooth and color and blue in the example "d1 marker circle line smooth color blue".
It is simple to change to a bar graph and include last year's measurements:

```
size 7 4
begin graph
   title  "Bar Graph"
   xtitle "Measurement"
   ytitle "Output"
   data   "year-2000.csv"
   data   "year-2001.csv"
   key pos tl
   bar d1,d2 fill red,blue
end graph
```



Adding min and max values on the axis commands is highly recommended because by default GLE won't start from the origin unless the data happens to be very close to zero. It is also difficult to compare graphs unless they all have the same axis ranges. More information about the graph module is available in Chapter 4.

Chapter 3

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 centimeters 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.

3.1 Graphics Primitives (a summary)

! *comment*
@*xxx*
abound *x y*
aline *x y* [arrow start] [arrow end] [arrow both] [curve $\alpha 1$ $\alpha 2$ *d1 d2*]
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 length *var*
begin name *name*
begin object *name*
begin origin
begin path [stroke] [fill *pattern*] [clip]
begin rotate *angle*
begin scale *x y*
begin table
begin tex
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
colormap *fct xmin xmax ymin ymax pixels-x pixels-y width height* [color] [palette *pal*]
curve *ix iy* [*x1 y1 x y x y ... xn yn*]*ex ey*
define marker *markername subroutine-name*
draw *name.point [arg1 ... argn] [name 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* [curve $\alpha 1$ $\alpha 2$ *d1 d2*]
local var_1, ..., var_n
margins *top bottom left right*
marker *marker-name* [*scale-factor*]
orientation *o*
papersize *size*
postscript *filename.eps width-exp height-exp*
print *string$* ...
psbbtweak
pscomment *exp*
rbezier *x1 y1 x2 y2 x3 y3*
return *exp*
reverse
rline *x y* [arrow end] [arrow start] [arrow both] [curve $\alpha 1$ $\alpha 2$ *d1 d2*]
rmove *x y*
save *objectname*
set alabeldist *d*
set alabelscale *s*
set arrowangle angle
set arrowsize size
set arrowstyle simple — filled — empty
set atitledist *s*
set atitlescale *s*
set background it c
set cap butt — round — square
set color *col*
set dashlen *dashlen-exp*
set fill *fill color/pattern*
set font *font-name*
set fontlwidth *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*
set pattern *fill pattern*
set texscale *scale* | *fixed* | *none*
set titlescale *s*
set ticksscale *s*
size *w h*
sub *sub-name parameter1 parameter2 etc*
tex *string* [name *xxx*] [add *val*]
text *unquoted-text-string*
write *string$* ...

3.2 Graphics Primitives (in detail)

! *comment*

Indicates the start of a comment. GLE ignores everything from the exclamation point to the end of the line. This works both in GLE scripts and in data files used in, e.g., graph blocks.

@*xxx*

Executes subroutine *xxx*.

abound *x y*

Update the current bounding box to include the point (x, y) without drawing anything. This command is useful in combination with 'begin box', 'begin name', etc., e.g., to add empty space to the box.

aline *x y* [arrow start] [arrow end] [arrow both] [curve $\alpha 1$ $\alpha 2$ *d1 d2*]

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.

If the curve option is given, then a Bezier curve is drawn instead of a line. The first control point is located at a distance $d1$ and angle $\alpha 1$ from the current point and the second control point is located at distance $d2$ and angle $\alpha 2$ from *(x,y)*.

amove *x y*

Changes the current point to the absolute coordinates *(x,y)*.

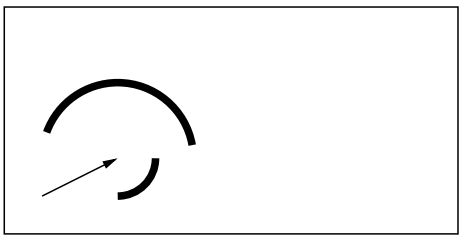
arc *radius a1 a2 [arrow end] [arrow start] [arrow both]*

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.

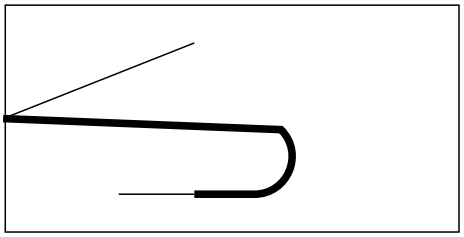
```
amove 0.5 0.5
rline 1 0.5 arrow end
set lwidth 0.1
arc 1 10 160
arc 0.5 -90  0
```



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.

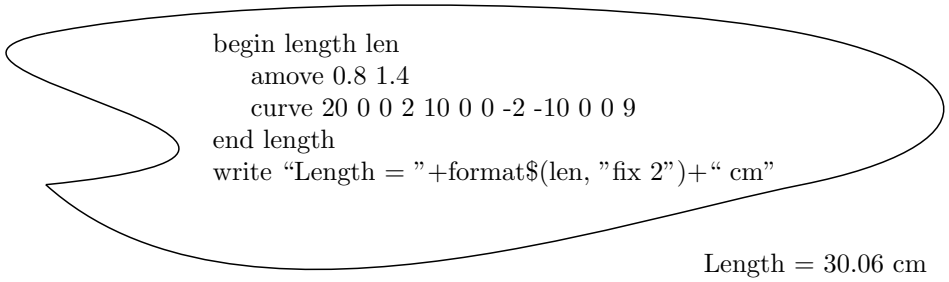
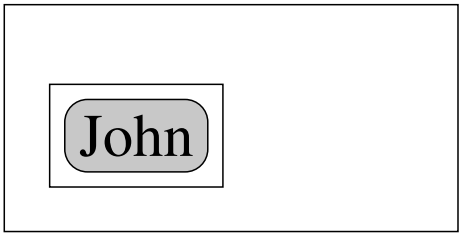


Figure 3.1: Compute the total length of a shape.

```
begin box add 0.2
   begin box fill gray10 add 0.2 round .3
      text John
   end box
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 length *var*
This block computes the total length of all the elements that are included in it and saves the result in the variable "*var*". See Fig. 3.1 for an example.

begin name *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 object *name* [*arg1, ..., argn*]
Declares a new object (sub-figure) that can be drawn later with the 'draw' command. Section 7.1.2 explains in detail how this command works and how it can be used.

Object blocks can have arguments if they are not defined inside a subroutine. Such object blocks are called 'static objects'; they behave similar to subroutines. Object blocks can also be defined inside a subroutine. In that case, they are called 'dynamic objects' and cannot have arguments. They may, however, refer to all arguments and local variables of the surrounding subroutine.

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.

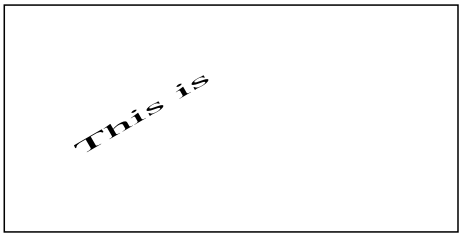
begin rotate *angle*
The coordinate system is rotated anti-clockwise about the current point by the angle *angle* (in degrees). For example, to draw a line of text running vertically up the page (as a Y axis label, say), type:

```
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.

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

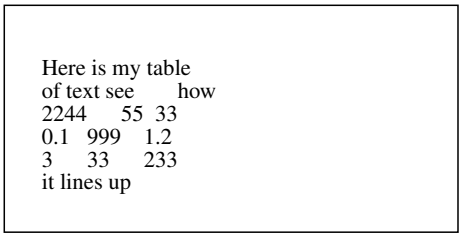


begin table

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.

```
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 LaTeX does, so multiple spaces are ignored and some characters have special meaning. E.g, `\ ^ _ & { }`

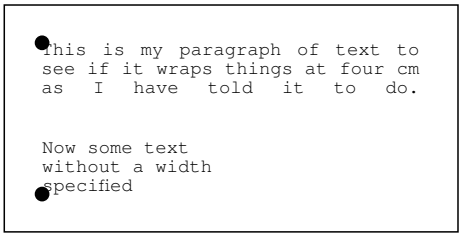
To include Greek characters in the middle of text use a backslash followed by the name of the character. E.g., `3.3\Omega S` would produce "3.3ΩS".

To put a space between the Omega and the S add a backslash space at the end. E.g., `3.3\Omega\ S` produces "3.3Ω S"

Sometimes the space control characters (e.g. `\:`) 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
```

```
set hei 0.25 just tl font tt
begin text width 5
  This is my paragraph of text to see
  if it wraps things at four cm as I have
  told it to do.
end text
...
begin text
  Now some text
  without a width
  specified
end text
```



There are several LaTeX like commands which can be used within text. The complete list can be found in Appendix A.3. A few examples are:

```
\ \' \v \u \= \^ \. \H \~ \'' Implemented TeX accents
^{} _{}                       Superscript, subscript
\\ \_                         Forced Newline, underscore character
\, \: \;                      0.5em, 1em, 2em space (em = width of the letter 'm')
\tex{expression}              Any LaTeX expression
\char{22}                     Any character in current font
\glass                        Makes move/space work on beginning of line
\rule{2}{4}                   Draws a filled in box, 2cm by 4cm
\setfont{rmb}                 Sets the current text font
\sethei{0.3}                  Sets the font height (in cm)
\setstretch{2}                Scales the quantity of glue between words
\lineskip{0.1}                Sets the default distance between lines of text
\linegap{-1}                  Sets the minimum required gap between lines
{\rm ...}, {\it ...}          Sets roman, and italic font
{\bf ...}, {\tt ...}          Sets bold, and typewriter (monospaced) font
\alpha, \beta, ...            Greek symbols
```

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

bezier *x1 y1 x2 y2 x3 y3*
: Draws a Bézier cubic section from the current point to the point *(x3,y3)* with Bézier cubic control points at the coordinates *(x1,y1)* and *(x2,y2)*. 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 *width*×*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.

bitmap_info *filename width height* [type *type*]
: Returns the dimensions in pixels of the bitmap in the output parameters *width* and *height*.

box *x y* [justify *jtype*] [fill *color*] [name *xxx*] [nobox] [round *val*]
: Draws a box, of width *x* and height *y*, with its bottom left corner at the current point. If the justify option is used, the box will be positioned relative to the specified point. E.g., TL = top left, CC = center center, BL = bottom left, CENTER = bottom center, RIGHT = bottom right, LEFT = bottom left. See set just for a description of justification settings.

  If a fill pattern is specified, the box will be filled. Remember that white fill is different from no fill pattern - white fill will erase anything that was inside the box.

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

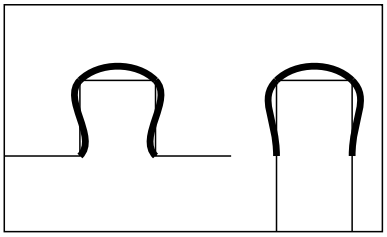
circle *radius* [fill *pattern*]
: Draws a circle at the current point, with radius *radius*. If a fill pattern is specified the circle will be filled.

closepath
: Joins the beginning of a line to the end of a line. I.e., it does an aline to the end of the last amove.

colormap *fct xmin xmax ymin ymax pixels-x pixels-y width height* [color] [palette *pal*]
: Draws a colormap of the function $fct(x, y)$, in which x ranges from *xmin* to *xmax*, and y ranges from *ymin* to *ymax*. The size of the colormap is *width* by *height* centimeter and the resolution is *pixels-x* by *pixels-y* pixels. A colormap is grayscale by default; it is drawn in color if the option *color* is given. In the latter case, it is possible to specify a palette subroutine *pal* mapping the range $0 \ldots 1$ to a range of colors. This command is similar to the colormap command in a graph block (Sec. 9.5).

curve *ix iy* [*x1 y1 x y x y ... xn yn*]*ex ey*
: Draws a curve starting at the current point and passing through the points *(x1,y1) ... (xn,yn)*, with an initial slope of *(ix,iy)* to *(x1,y1)* and a final slope of *(ex,ey)*. All the vectors are relative movements from the vector before.

```
amove 1 1
curve 1 0 0 1 1 0 0 -1 1 0
amove 3.6 1
curve 0 1 0 1 1 0 0 -1 0 -1
```



define marker *markername subroutine-name*
: This defines a new marker called *markername* which will call the subroutine *subroutine-name* whenever it is used. It passes two parameters, the first is the requested size of the marker and the second is a value from a secondary dataset which can be used to vary size or rotation of a marker for each point plotted.

To define a character from the postscript ZapDingbats font as a marker you would use, e.g.

```
sub subnamex size mdata
   gsave                              ! save font and x,y
   set just left font pszd hei size
   t$ = "\char{102}"
   rmove -twidth(t$)/2 -theight(t$)/2  ! centers marker
   write t$
   grestore                           ! restores font and x,y
end sub
```

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
```

draw *name.point [arg1 ... argn] [name name]*
: Draws a named object block that has been previously defined using a "begin/end object" (p. 10) construct. The object is drawn such that the point indicated by the first argument of the draw command appears at the current position. The point can be any (hierarchically) named point on the object and may include the justify options defined for the join command (p. 15).

If the object block has parameters (similar to a subroutine) then these parameters can be given as *arg1 ... argn*.

The "draw" command names the object using the same name as the name of the object block by default. An alternative name can be supplied using its "name" option.

See Sec. 7.1.2 for a detailed explanation of this command with examples.

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*

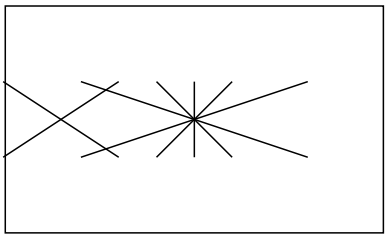
The for ... next structure lets you repeat a block of statements a number of times.

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 x = 1 to 4 step 0.5
   amove x 1
   aline 5-x 2
next x
```



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*

Includes the GLE script "filename" into the current script. This is useful for including library scripts with subroutines. GLE searches a number of predefined directories for include files. By default, this includes the current directory and the "lib" or "gleinc" subdirectory of the root directory (GLE_TOP) of your GLE installation. The latter includes a number of subroutine files that are distributed with GLE (Table 3.1). Additional include directories can be defined by means of the environment variable GLE_USRLIB.

Table 3.1: Include files distributed with GLE.

barstyles.gle	Defines additional styles for bar plots.
color.gle	Defines functions for working with colors.
colors-gle-4.0.12.gle	Redefines all colors defined in GLE 4.0.12 and before.
contour.gle	Subroutines for drawing contour plots
electronics.gle	Subroutines for drawing electrical circuits
ellipse.gle	Draw text in an ellipse
feyn.gle	Subroutines for drawing Feynmann diagrams
graphutil.gle	Subroutines for drawing graphs
piesub.gle	Pie chart routines
polarplot.gle	Polar plotting routines
shape.gle	Drawing various shapes
simpletree.gle	Draw simple trees
stm.gle	Add labels to images
ziptext.gle	Draw zipped text

join *object1.just sep object2.just* [curve $\alpha 1$ $\alpha 2$ *d1 d2*]

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: `.br` (bottom right), `.bl` (bottom left), `.bc` (bottom centre), `.tr` (top right), `.tc` (top centre), `.tl` (top left), `.cr` (centre right), `.cc` (centre centre), and `.cl` (centre left). In addition, `.v` and `.h` can be used to draw vertical or horizontal lines connecting to the object, `.c` for drawing a line connecting to e circle or ellipse, and `.box` for drawing a line to a rectangle. Fig. 3.4 shows examples of the different cases.

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 centers of the two objects (clipped at the edges of the rectangles which define the objects). This is the same as using the `.box` qualifier on both objects.

The curve option is explained with the aline command. Fig. 3.4 shows an example where the "join" command is used with the curve option.

Sec. 7.1.1 contains several examples of joining objects.

local var_1, ..., var_n

Defines a local variable inside a subroutine. It is possible to initialize the variable to a particular value with, e.g., '`local x = 3`', which defines the local variable 'x' and assigns it the value 3. You can also define several local variables at once, e.g., '`local x, y`' defines the local variables 'x' and 'y'.

margins *top bottom left right*

This command can be used to define the page margins. Margins are only relevant for making full-page figures (using the -fullpage command line option). See also the "papersize" command.

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 Fig. 3.2.

```
set just lc
amove 0.5 2.5
marker diamond 1
```

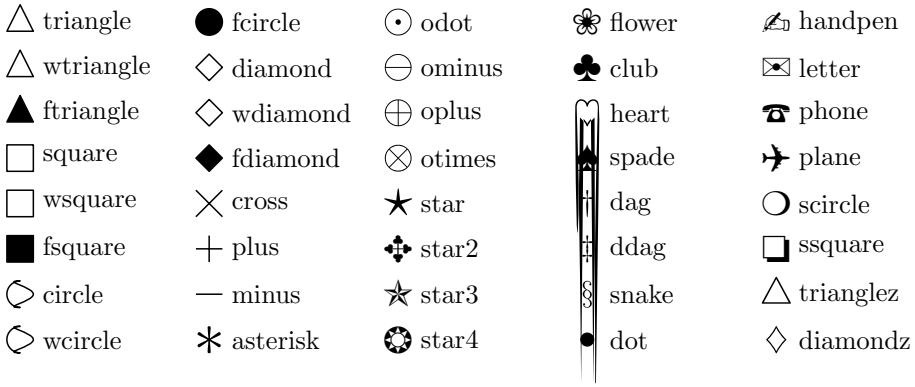


Figure 3.2: All markers supported by GLE. (The names that start with "w" are white filled.)

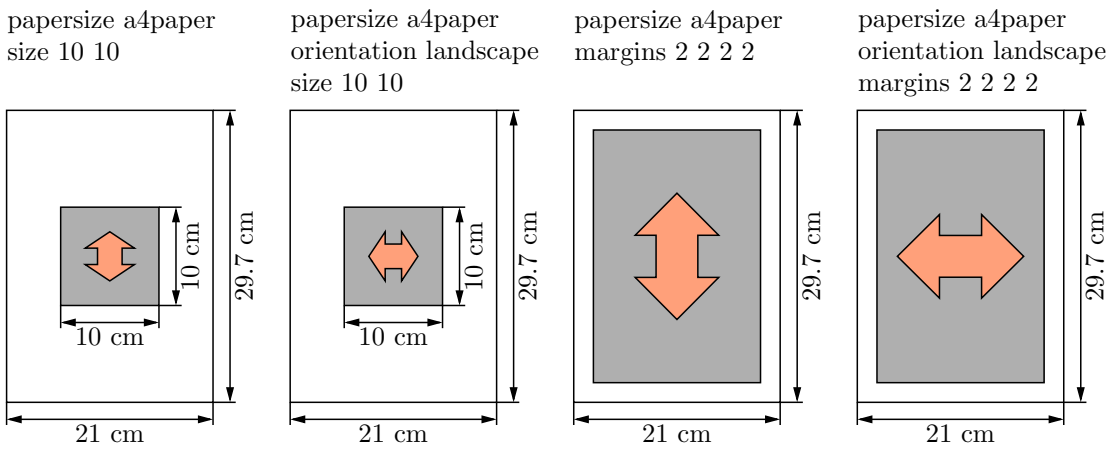


Figure 3.3: Result of different combinations of the commands "papersize", "margins", "size", and "orientation" for fullpage graphics (gle -fullpage figure.gle).

```
rmove 0.6 0; text Diamond
amove 0.5 2
marker triangle 1
rmove 0.6 0; text Triangle
...
```

orientation *o*

Sets the orientation of the output in full-page mode. Possible values are "portrait" and "landscape". Fig. 3.3 illustrates these two cases.

papersize *size*

papersize *width height*

Sets the paper size of the output. This is used only when GLE is run with the option "-fullpage" or when the PostScript output device is used (i.e., "-d ps"). The command either takes one argument, which should be one of the predefined paper size names or two numbers, which give the width and height of the output measured in cm. The following paper sizes are known by GLE: a0paper, a1paper, a2paper, a3paper, a4paper, and letterpaper.

If a "size" command is given in the script, then the output is drawn centered on the page. If no size command is included in the script, then the output will appear relative to the bottom-left corner of the page, offset by the page margins (see "margins" command). Fig. 3.3 illustrates these two cases.

The paper size can also be set in GLE's configuration file (Sec. 7.5).

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.

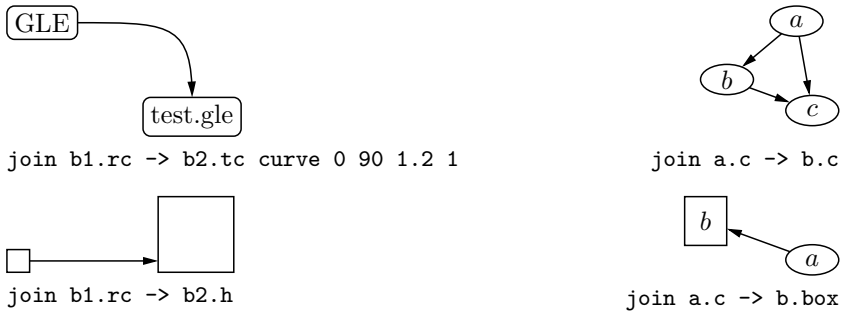


Figure 3.4: Different ways of joining objects.

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).

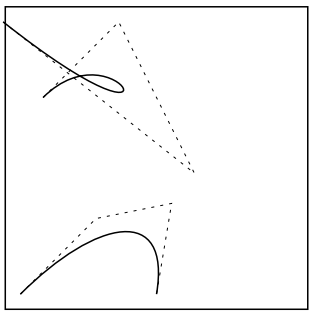
This command must appear before the first size command in the GLE file.

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   0.5 2.8
rbezier 1   1   2 -1  3   1
amove   0.2 0.2
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.

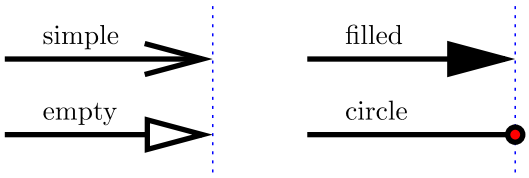
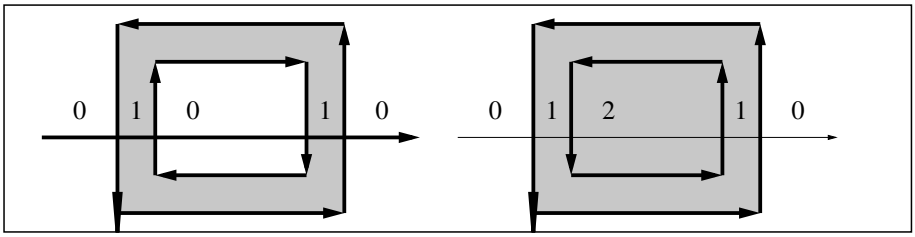


Figure 3.5: Different arrow tip styles.



rline x y [arrow end] [arrow start] [arrow both] [curve $\alpha 1$ $\alpha 2$ *d1 d2*]
: 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.

  The curve option is explained with the aline command.

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 alabeldist d
: The spacing between the graph axis labels and the axis is set to d.

set alabelscale s
: The graph axis label font size is set to '`alabelscale`' times '`hei`'.

set arrowangle angle
: Sets the opening angle of the arrow tips. (Actually, half of the opening angle.)

set arrowsize size
: Sets the length of the arrow tips in centimeter.

set arrowstyle simple — filled — empty
: Sets the style of the arrow tips. There are three pre-defined styles: simple, filled, and empty (See Fig. 3.5).

  It is also possible to create user-defined arrow tip styles. To do so, create a subroutine 'arrow_xxxx langle aangle asize', with xxxx the name of the new style. The parameter langle is the angle of the line on which the arrow tip is to be drawn and the parameters aangle and asize are the current values of the settings arrowangle and arrowsize. The user-defined style can be enabled, in the same way as the built-in ones, with 'set arrowstyle xxxx'. Fig. 3.5 shows the three predefined styles and a user-defined tip style that is defined by the following subroutine:

```
sub arrow_circle langle aangle asize
   circle 0.1 fill red
end sub
```

  More complex examples of user-defined arrow styles can be found in the GLE example repository.

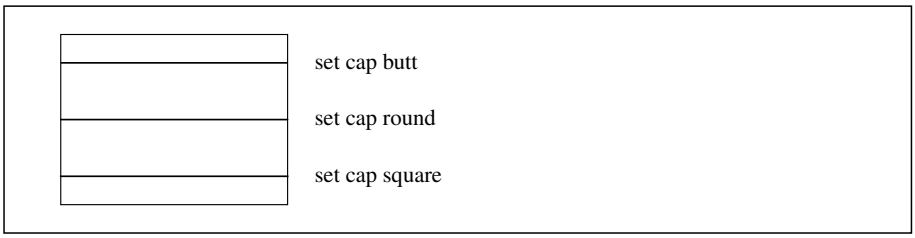
set atitledist s
: The spacing between the graph axis title and the axis labels is set to d.

Figure 3.6: Examples of setting the drawing color.

set atitlescale *s*
: The graph axis title font size is set to '`atitlescale`' times '`hei`'.

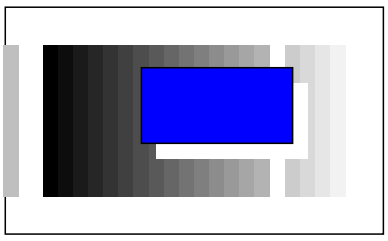
set background it c
: Set the background color for a pattern fill to *c*. (See 'set fill'.) Note that "set background" must come after "set fill" because "set fill" resets the background color to the specified color.

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. GLE supports all SVG/X11 standard color names. These are listed in Appendix A.8, and include the following: black, white, red, green, blue, cyan, magenta, yellow, gray10, gray20, ..., gray90. It is also possible to specify a gray scale as a real number with 0.0 = black and 1.0 = white. Colors can also be set using the HTML notation, e.g., #FF0000 = red. Finally, the functions rgb(red,green,blue) and rgb255(red,green,blue) may be used to create custom colors. Fig. 3.6 gives some examples.

```
mm$ = "blue"
amove 0.5 0.5
for c = 0 to 1 step 0.05
   box 0.2 2 fill (c) nobox
   rmove 0.2 0
next c
amove 2 1
box 2 1 fill white nobox
rmove -0.2 0.2
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 fill *fill color/pattern*
: Sets the color or pattern for filling shapes. This command works in combination with shapes such as circles, ellipses, and boxes. If the argument is a color, then shapes are filled with the given color (see "set color"). If it is a pattern, then the shapes are painted with the given pattern in black ink. Fig. 3.7 lists a number of pre-defined patterns. To paint a shape in a color different from black, first set the color, then the pattern. That is,

```
set fill red
set pattern shade
set background yellow
box 2 2
```

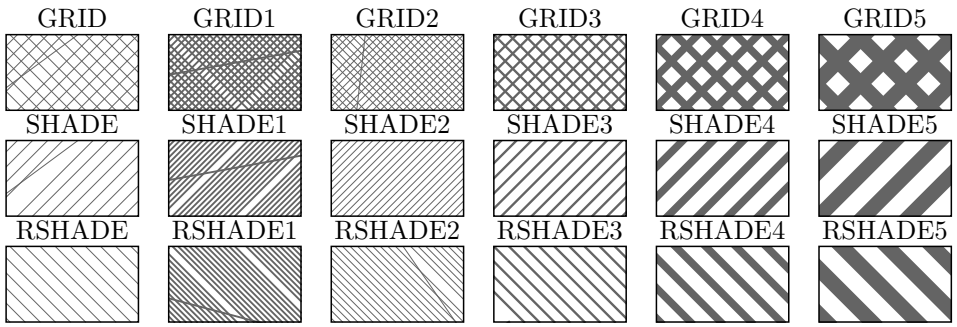


Figure 3.7: Patterns for painting shapes.

will draw a box and paint is using the shade pattern and red ink on a yellow background. To draw shapes that are not filled, use the command "set fill clear". That is,

```
set fill clear
box 2 2
```

will draw an empty box.

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

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

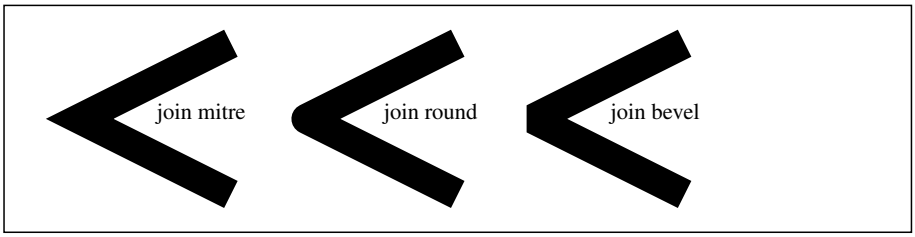
set fontlwidth *line-width*
This sets the width of lines to be used to draw the stroked (Plotter fonts) on a PostScript printer. This has a great effect on their appearance.

```
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.

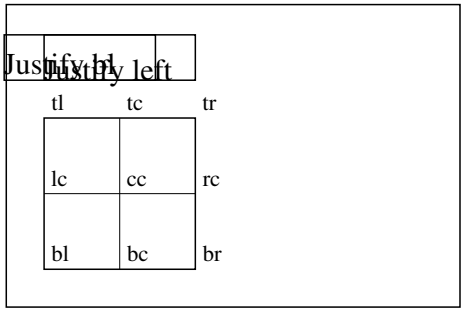
The default value of "hei" is 0.3633 (to mimic the default height of LaTeX expressions).

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.



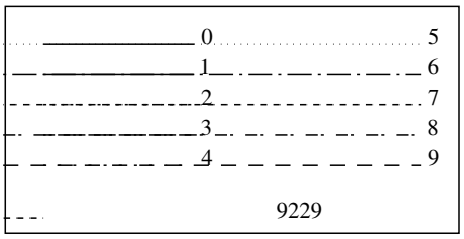
set just left | center | right | tl | etc...
Sets the justification which will be used for text commands.

```
amove 0.5 3
set just left
box 1.5 0.6
text Justify left
rmove 2 0
set just bl
box 1.5 0.6
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 just left
for z = 0 to 4
   set lstyle z
   rline 2 0
   rmove 0.1 0
   write z
   rmove -2.1 -0.4
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.

set pattern *fill pattern*
Specifies the filling pattern. A number of pre-defined patterns is listed in Fig. 3.7. See the description of "set fill" for more information. Note that "set pattern" must come after "set fill" because "set fill" resets the pattern to solid.

set texscale *scale* | *fixed* | *none*
This setting controls the scaling of LaTeX expressions (Sec. 7.2): '`scale`' scales them to the value of '`hei`', '`fixed`' scales them to the closest LaTeX default size to '`hei`', and '`none`' does not scale them. With '`none`', the font size in your graphics will be exactly the same as in your main document.

set titlescale *s*
The graph title font size is set to '`titlescale`' times '`hei`'.

set ticksscale *s*
The size of the graph axis ticks is set to '`ticksscale`' times '`hei`'.

size *w h*
Sets the size of GLE's output to *w* centimeter wide by *h* centimeter tall.

This command usually appears at the top of a GLE script. That is, only commands that do not generate output can precede the 'size' command. For example, the 'include' command, subroutine definitions, and assignments to variables can appear before the 'size' command. Commands like 'aline', on the other hand, should appear after the 'size' command.

It is possible to omit the size command. In that case, the size of the output is determined by the 'pagesize' command (see Fig. 3.3).

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 re-entrant.

```
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*] [add *val*]

Draw a LaTeX expression at the current point using the current value of 'justify'. See Sec. 7.2 for more information. Using the name option, the LaTeX expression can be named, just like a box. The size of the virtual named box can be increased with the add option.

text *unquoted-text-string*

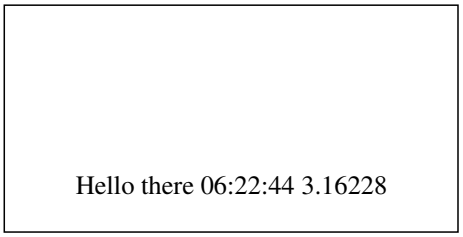
This is the simplest command for drawing text. The current point is unmodified after the text is drawn so following one text command with another will result in the second line of text being drawn on top of the first. To generate multiple lines of text, use the begin text . . . end text construct.

```
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.

```
a$ = "Hello there "
xx = sqrt(10)
t$ = time$()
c$ = a$+t$
write a$+t$ xx
```



The built in functions sqrt() and time$() are described in Appendix A.2.

Chapter 4

The Graph Module

A graph should start with begin graph and 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 1000, e.g. d1
The name dn can be used to define a default for all datasets. Many graph commands described below start with d*n*. 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.

4.1 Graph Commands (a summary)

center
colormap *fct pixels-x pixels-y* [*color*] [*invert*] [*zmin* z_1] [*zmax* z_2] [*palette pal*]
data *filename* [*d1 d2 d3 ...*] [*d1=c1,c3 ...*] [ignore *n*] [comment *char*]
discontinuity threshold *t*
dn [deresolve m] [average] line
dn err *d5* errwidth *width-exp* errup *nn%* errdown *d4*
dn herr *d5* herrwidth *width-exp* herrleft *nn%* herrright *d4*
dn key *"Dataset title"*
dn line [impulses] [steps] [fsteps] [hist] [bar]
dn lstyle *line-style* lwidth *line-width* color *col*
dn marker *marker-name* [color *c*] [msize *marker-size*] [mdata *dn*] [mdist *dexp*]
dn nomiss
dn smooth — smoothm
dn svg_smooth [m]
dn xmin *x-low* xmax *x-high* ymin *y-low* ymax *y-high*
dn [x2axis] [y2axis]
dn file "all.dat,xc,yc" [marker mname] [line]
d[*i*] ...
draw call
fullsize
hscale h
key pos *tl* nobox hei *exp* offset *xexp yexp*
let ds = *exp* [from *low*] [to *high*] [step *exp*] [where *exp*]
let ds = *x-exp, y-exp* [from *low*] [to *high*] [step *exp*] [where *exp*]
let dn = [routine] dm [options]
let ds = hist dm [from *x1*] [to *x2*] [bins *n*] [step *n*]
let ds = ... [nsteps *n*]
let ds = ... [range *dn*]
math
nobox
scale *h v*

scale auto
size *x y*
title *"title"* [hei *ch-hei*] [color *col*] [font *font*] [dist *cm*]
vscale v
x2labels on
xaxis — yaxis — x2axis — y2axis
xaxis angle α
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 ftick *x0* dticks *distance*
xaxis off
xaxis shift *cm-exp*
xaxis symticks
xlabels font *font-name* hei *char-hei* color *col*
xnames *"name" "name" ...*
xnames from *dx*
xnoticks *pos1 pos2 pos3 ...*
xplaces *pos1 pos2 pos3 ...*
xside color *col* lwidth *line-width* off
xsubticks lstyle *num* lwidth *exp* length *exp* on off
xticks lstyle *num* lwidth *exp* length *exp* off
xtitle *"title"* [hei *ch-hei*] [color *col*] [font *font*] [dist *cm*] [adist *cm*]
y2title *"text-string"* [rotate]
yaxis negate
bar *dx,...* dist *spacing*
bar *dn,...* fill *f* pattern *p*
bar *dx,...* from *dy,...*
bar *dn,...* horiz
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*

4.2 Graph Commands (in detail)

center
: Centers the graph (including the title and axis labels) in the graph box. (The command 'scale auto' implicitly performs 'center'.)

colormap *fct pixels-x pixels-y* [*color*] [*invert*] [*zmin* z_1] [*zmax* z_2] [*palette pal*]
: The colormap command is discussed in Section 9.5.

data *filename* [*d1 d2 d3 ...*] [*d1=c1,c3 ...*] [ignore *n*] [comment *char*]
: 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 1000, e.g. d1 or d4. Up to 1000 datasets may be defined.

  From a file with 3 columns the command 'data "data.csv"' 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

```
begin graph
  xtitle "Age"
  ytitle "Weight"
  data "age.csv"
  key pos br compact
  d1 line color red marker circle
  d2 line color blue marker triangle
  d3 line color green marker square
end graph
```

```
Age, John, Mary, Ken
15,  73,   65,   77
20,  80,   68,   77
30,  82,   76,   80
40,  85,   77,   82
```

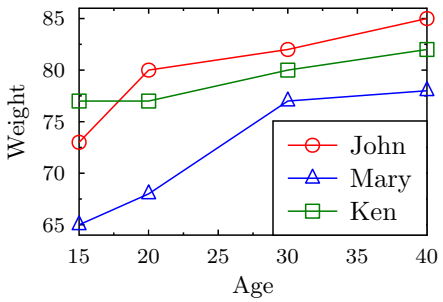


Figure 4.1: Line graph with key taken from the column labels in the first data row. Left: the graph block; middle: the dataset "age.csv"; right: the resulting graph.

dataset 2 (d2). The next data command would use dataset 3 (d3).

Such a data file looks like this:

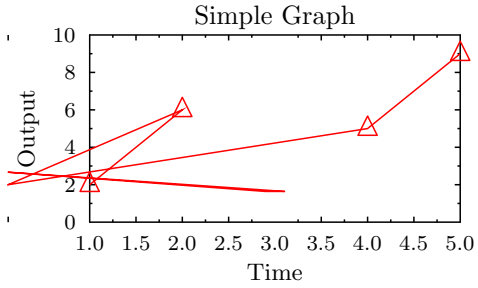
```
1, 2.7, 3
2, 5,   *
3, 7.8, 7
4, 9,   4
```

The first point of dataset **d1** would then be (**1, 2.7**) and the first point of dataset **d2** would be (**1, 3**). The data values can be space, tab, comma, or semi-colon separated.

Missing values can be indicated with "*", "?", "–", or ".".

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

```
size 7 3.5
begin graph
   size   6 3
   title  "Simple Graph"
   xtitle "Time"
   ytitle "Output"
   data   "data.csv"
   d1 line marker triangle color red
end graph
```



Comments: Comments can be included with the symbol "!". All characters from "!" until the end of the line of the data file are ignored. It is possible to change the symbol that indicates a comment with the option 'comment'. E.g., with 'data "data.csv" comment #', lines starting with # will be treated as comments.

Ignore header: The option ignore n makes GLE ignore the first n lines of the data file. This is useful if the first n lines do not contain data.

Auto key: If the first row of a data file does not contain actual data but instead contains column labels, then these labels are used by GLE to create a key for the graph (Chapter 5). GLE automatically detects this case by checking if all fields in the first row are valid numbers or not. If not, then GLE assumes that the first row contains column labels. Column labels that include a space or that could be incorrectly interpreted as a number should be double quoted. Fig. 4.1 illustrates this with an example.

Auto x-labels: If the first column of a data file does not contain numeric data, but instead contains symbolic labels, then these labels are used to label the horizontal axis. For example, if the data file contains

```
Mon, 1
Tue, 4
Wed, 3.5
Thu, 2
Fri, 1
```

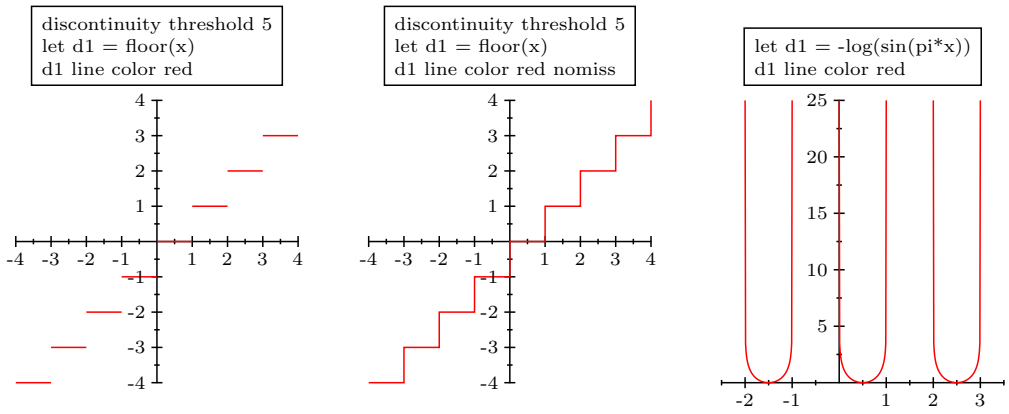


Figure 4.2: An example of the auto-discontinuity detection feature.

```
Sat, 5
Sun, 4
```

then GLE creates an x-axis with one label for each day of the week, similar to that of the graph in Fig. 4.16. See also the 'xnames' command for more details on how to add labels to an axis.

GLE can also read GZIP compressed data files. If the data file name ends in ".gz", then GLE will assume it is GZIP compressed and read it accordingly.

discontinuity threshold *t*

GLE can automatically detect discontinuities in graphs. To enable this feature, add "discontinuity threshold *t*" to the graph block. The value of t is the percentage of the axis range that the graph needs to change in one step in order to be detected as a discontinuity. Fig. 4.2 plots the "floor" function as an example.

The discontinuity detection feature inserts a missing value at the position of each discontinuity. This leads to gaps in the curve (Fig. 4.2, left). These can be disabled by using the "nomiss" keyword (Fig. 4.2, middle).

dn [deresolve m] [average] line

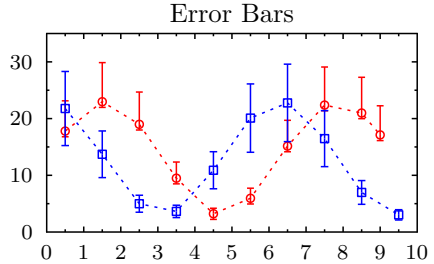
The 'deresolve' option sub-samples a dataset. Given the parameter $m > 1$, it keeps only 1 out of every m points (starting with the first point). If the option 'average' is given, then it will compute the average of the y-values of every window of m points. This average value will be plotted at the middle (x-value) of the window. The 'deresolve' option never removes the first and last point in a dataset if it is used in conjunction with 'dn line'.

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
   title "Error Bars"
   dn lstyle 2 msize 1.5
   d1 marker circle errup 30% errdown 1
   d2 marker square err   30% errwidth .1
end graph
```



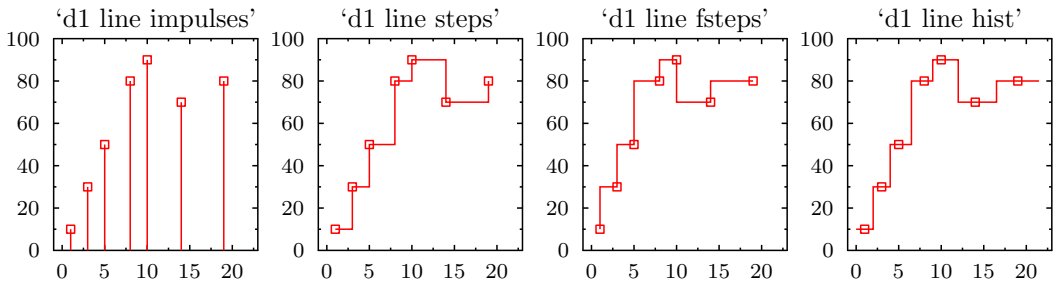


Figure 4.3: The impulses, steps, fsteps, and hist options of the line command.

dn herr *d5* herrwidth *width-exp* dn herrleft *nn%* herrright *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 5) to draw more complex keys.

dn line [impulses] [steps] [fsteps] [hist] [bar]
: 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 options impulses, steps, fsteps, hist, and bar draw lines as shown in Figure 4.3.

  - 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.
  - bar: similar to 'hist', but now also separates the bins with vertical lines.

dn lstyle *line-style* lwidth *line-width* color *col*
: These qualifiers are all fairly self explanatory. See the lstyle command in Chapter 3 (Page 21) for details of specifying line styles.

dn marker *marker-name* [color *c*] [msize *marker-size*] [mdata *dn*] [mdist *dexp*]
: 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, ...). The marker's color can be specified with the 'color' option.

  Markers can also be drawn using a user-defined subroutine (See the define marker 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 and date dependent 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.

  The 'mdist' option can be used to specify the distance between the markers on a curve. This can be used to add markers to a plot of a continuous function. See Fig. 5.3 for an example.

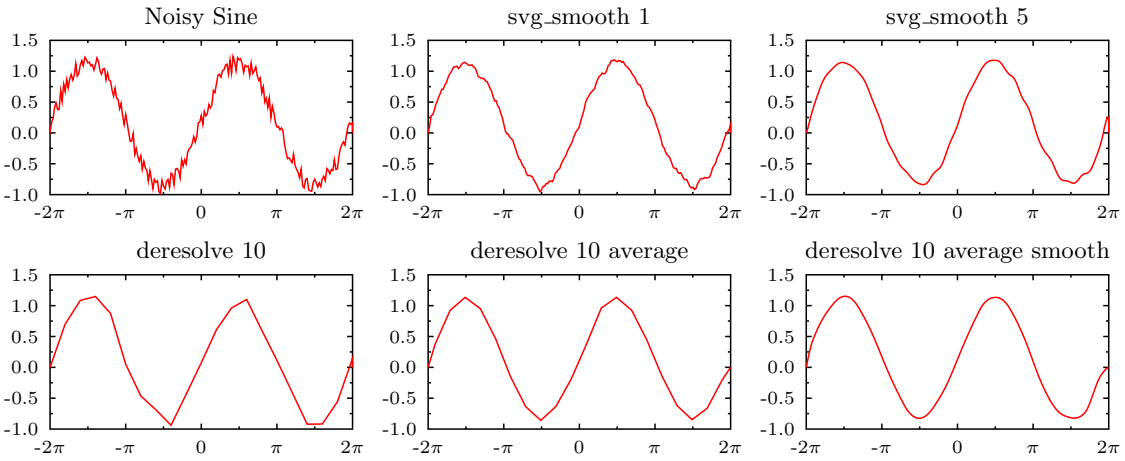
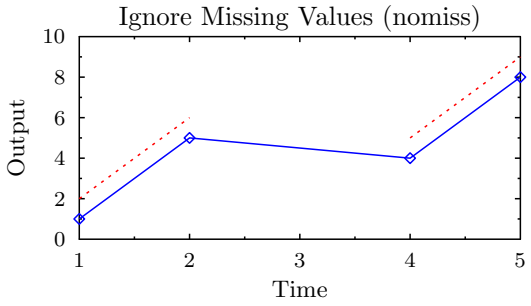


Figure 4.4: Various smooth options.

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.

```
begin graph
   title  "Ignore missing values (nomiss)"
   xtitle "Time"
   ytitle "Output"
   data   "tut.dat"
   d1 lstyle 2
   d2 nomiss lstyle 1 marker diamond msize .2
end graph
```



dn [smooth] [smoothm] line

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 [svg_smooth] [m] line

The option svg_smooth performs a quadratic or cubic and 7 point Savitsky Golay smoothing on the data. The parameter m specifies the number of iterations of smoothing, that is, the smoothing algorithm is run m times on the dataset. Fig. 4.4 shows an example.

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 labeled from *x-low* to *x-high* (regardless of how the axis is actually labeled). A point in the dataset at X = *x-low* will appear on the left hand edge of the graph.

dn [x2axis] [y2axis]

Sometimes one needs to draw two or more curves on the same graph that have different scales or of which the values are measured in different units. In such cases, some of the curves may be associated to the x2axis and/or the y2axis.

The example in Fig. 4.5 illustrates this for the parabola $y = x^2$ and the sine function $y = \sin x$. The former is scaled to the x/y axis as usual and the latter is scaled to the x2/y2 axis.

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

The 'file' option specifies a file from which the dataset is to be loaded. This option is an alternative to the "data" command (p. 24).

```
begin graph
   let d1 = x^2 from -2 to 2
   let d2 = sin(x) from 0 to 4*pi
   d1 line color red key "$x^2$"
   d2 x2axis y2axis line color blue key "$\sin(x)$"
   x2axis format pi min 0 max 4*pi dticks pi
end graph
```

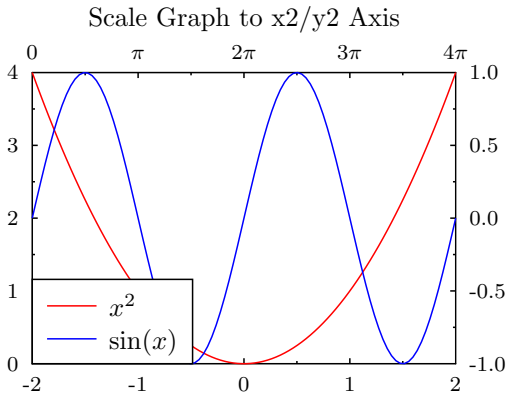


Figure 4.5: A parabola scaled to the x/y axis and the sine function scaled to the x2/y2 axis.

```
begin graph
   ...
   for alpha = 1 to 10 step 2
      let d[alpha] = sqrt(alpha*x) from 0 to 10
      d[alpha] line color rgb(alpha/10,0,0)
   next alpha
end graph
...
for alpha = 1 to 10 step 2
   amove xg(xgmax)+0.1 yg(sqrt(alpha*xgmax))
   write "$\alpha = "+num$(alpha)+"$"
next alpha
```

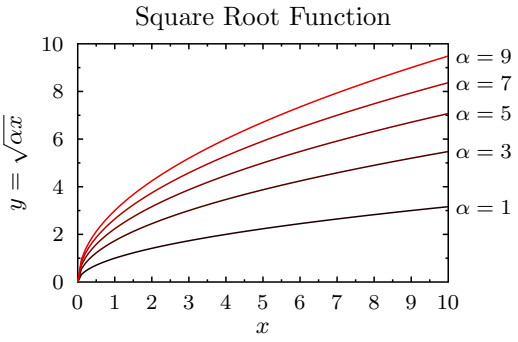


Figure 4.6: For-next loops in graph blocks and the use of "d[i]".

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...

The file option also accepts variables in place of the file name, e.g.:

```
x$ = "test.dat,2,3"
d1 file x$ line color red
```

d[i] ...
: A data set identifier "di" can also be written using the array notation "d[i]". Any valid expression that results in an integer can be used inside d[...]. This is useful if you want to select a data set based on the result of an expression.

  If-then-else, for-next loops, and other control constructs can be used insdide a graph block. These can be combined with the d[i] notation to draw many similar functions. See Fig. 4.6 for an example that draws the functions $y = \sqrt{\alpha x}$ with α an integer constant.

draw call
: Executes subroutine 'call' while drawing the graph. The call is drawn in the current layer (See "begin layer"). The output is clipped to the graph window and the subroutine can use the functions 'xg()', 'yg()', and variables 'xgmin', 'ygmin', etc. This is useful for drawing a custom graph background (Fig. 4.7) or for defining a custom graph type (Section 4.5.1).

fullsize
: This is equivalent to scale 1 1, noborder. It makes the graph size command specify the size and position of the axes instead of the size of the outside border. See Fig. 4.12 (right) for an example.

hscale h
: Sets the length of the xaxis to h times the size of the graph box (default is 0.7). h can also be set to 'auto'. See scale for more details.

```
sub background
   amove xg(xgmin) yg(ygmin)
   local wd = xg(xgmax)-xg(xgmin)
   local hi = yg(ygmax)-yg(ygmin)
   colormap y 0 0 0.8 1 1 200 wd hi
end sub

begin graph
   ...
   d1 line marker wdiamond color steelblue
   d2 line marker wcircle color green
   begin layer 150
      draw background
   end layer
end graph
```

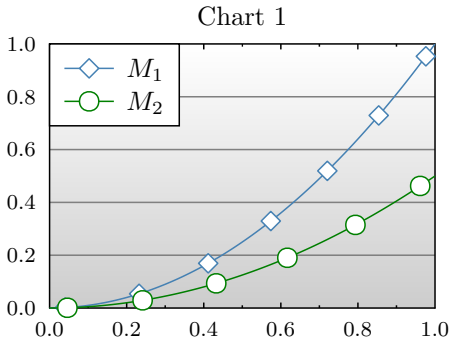


Figure 4.7: Using the "draw" command to draw a graph background.

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.

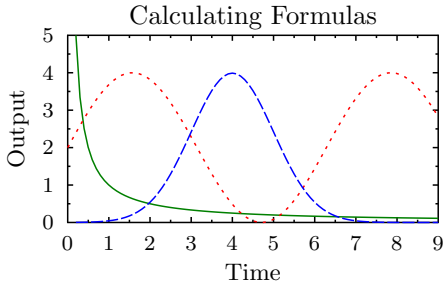
let ds = *exp* [from *low*] [to *high*] [step *exp*] [where *exp*]
: This command defines a new dataset as the result of an expression on the variable x over a range of values. For example:

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

NOTE: The lack of spaces inside the expression are necessary.

Here are some further examples:

```
begin graph
   ...
   let d1 = 1/x from 0.2 to 10
   let d2 = sin(x)*2+2 from 0 to 10
   let d3 = 10*(1/sqrt(2*pi))* &
            exp(-2*(sqr(x-4)/sqr(2)))) &
            from 0.2 to 10 step 0.1
   dn line
   d2 lstyle 2 color red
   d3 lstyle 3 color blue
end graph
```



The let command also allows the use of other datasets. E.g., to generate an average of two datasets:

```
data "file.csv" d1 d2
let d3 = (d1+d2)/2
```

More precisely, this command creates a dataset with as x-values the union of the x-values from d1 and d2, and as y-values the average of the y-values of d1 and d2. That is, the let command first computes the set of x-values by taking the union of the sets of x-values of all datasets that appear in the expression together with the x-values generated by the from/to/step construct. Then it iterates over these x-values. In each iteration, it assigns the x-value to the variable 'x', and, for each dataset included in the expression, it assigns its corresponding y-value to the dataset identifier. Then it evaluates the given expression *exp* and adds the resulting point (x,*exp*) to the target dataset. If the let expression includes more than one dataset, and the x-ranges of these datasets are different, then linear interpolation is used to compute the missing y-values.

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. The "from", "to", and "step" parameters are optional. The values of "from" and "to" default to the horizontal axis' range.

This command can also be used to modify the values in a data set, e.g., 'let d1 = 2*d1', will multiply all y-values in dataset d1 by 2.

The option 'where *exp*' can be used to select values from a dataset, e.g., 'let d1 = d2 where ((x > 10) and (x < 20))' will select all points from d2 for which the x-value is between 10 and 20; 'let d1 = d2 where d2 < 10' will select all points for which the y-value is below 10.

let ds = *x-exp, y-exp* [from *low*] [to *high*] [step *exp*] [where *exp*]

This syntax for the 'let' command is similar to the previous one, but now two expressions can be given: *x-exp* is used to compute the x-values of the points in the target dataset 'ds' and *y-exp* is used to compute the y-values. The parameter that is modified by the from/to/step construct is still the variable 'x'.

This syntax can be used to perform transformations on both the x and the y-values of the points in a dataset. For example,

```
let d1 = 2*x, d1+4
```

will multiply the x-values of d1 by 2 and add 4 to the y-values.

This syntax can also be used to define datasets that are not functions. The following example defines a circle:

```
let d1 = sin(x), cos(x) from 0 to 2*pi
```

let *dn* = [routine] *dm* [options] [*slopevar*] [*offsetvar*] [*rsqvar*]

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*.

The following routines are available :

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

The value for a is stored in '*slopevar*' and the value for b is stored in '*offsetvar*'. The r^2 value of the fit is stored in '*rsqvar*'. Note that these variables are optional.

The following options are available :

- `from` *xmin* `to` *xmax* The range of the data in *dn* extends from the *xmin* to *xmax* as specified by the user.
- `step` *xstep* Specifies the x-resolution of the fitted equation. Similar to the `step` option of the `let` command.
- `rsq` *var* The r^2 value of the fit will be stored in *var*.
- `xmin` *x1*, `xmax` *x2*, `ymin` *y1*, `ymax` *y2* Only use data points from dm in the given window to fit the equation. That is, only data points (x, y) from dm are used for which $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$.
- `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*.

let *dn* = fit *dm* with *eqn* [options]

Fit the coefficients of a given equation so that it best fits the data in dataset dm. Fig. 4.9 shows an example. The equation to fit is given by the '`with`' option. In this example, it is $a\sin(bx) + cx^2 + d$. GLE will search for values for the coefficients a, b, c, and d such that the given equation fits dm best. Note that all used coefficients must be initialized to zero before the graph block (see figure).

The fit command has the same options as the linfit command. In addition, it has the following settings.

```
slope = 0; offs = 0; rsquare = 0

set texlabels 1
begin graph
   title "Linear fit"
   xtitle "$x$"
   ytitle "$y = ax + b$"
   data "data.csv"
   let d2 = linfit d1 from 0 to 10 slope offset rsquare
   d1 marker circle color blue
   d2 line color red
end graph

begin key
   pos br nobox
   text "$y = "+format$(slope,"fix 2")+"x + "+format$(offset,"fix 2")+"$"
   text "$r^2 = "+format$(rsquare,"fix 2")+"$"
end key
```

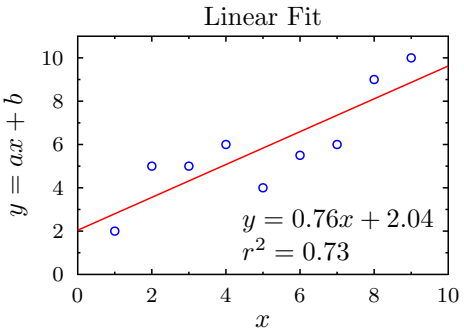


Figure 4.8: Fitting linear equations 'let d2 = fitlin d1'.

- `with` *eqn* Gives the equation to fit.
- `eqstr` *strvar$* Sets the string variable in which the string representation of the fitted equation is to be stored.
- `format` *fmt$* Sets the numeric format to use while converting the fitted equation into its string representation. See the documentation of format$ on page 52 for a description of the syntax.

let ds = hist dm [from *x1*] [to *x2*] [bins *n*] [step *n*]
: Computes a histogram for the values in 'dm' and store the result in 'dn'. E.g., if the file 'normal.csv' contains a single column with samples from the standard Gaussian distribution, then the graph block

```
begin graph
   ...
   data "normal.csv"
   let d2 = hist d1 step 0.5
   d2 line bar color red
end graph
```

will result in the histogram shown in Fig. 4.10.

The option 'bins' specifies the number of bins in the histogram. Alternatively, the option 'step' can be used to specify the bin size.

let ds = ... [nsteps *n*]
: The 'nsteps' options is an alternative to the 'step' option of the 'let' command. It specifies the total number of steps rather than the step width. The default value for 'nsteps' is 100.

let ds = ... [range *dn*]
: Takes the x-values for this 'let' expression from dataset 'dn'. This is useful if you need to define a function for the same x-values as the ones you have in a given dataset.

math
: Use this option to create a math mode graph, with the axis crossing at point $(0, 0)$. Fig. 4.11 shows an example. The corresponding GLE code is as follows:

```
begin graph
   math
   title "f(x) = sin(x)"
   xaxis min -2*pi max 2*pi ftick -2*pi dticks pi format "pi"
   yaxis dticks 0.25 format "frac"
   let d1 = sin(x)
   d1 line color red
end graph
```

```
a = 0; b = 0; c = 0; d = 0; r = 0

set texlabels 1
begin graph
   xtitle "$x$"
   ytitle "$f(x)$"
   title "$f(x) = a\sin(bx)+cx^2+d$"
   data "data.csv"
   let d2 = fit d1 with a*sin(b*x)+c*x^2+d rsq r
   d1 marker circle color blue
   d2 line color red
end graph

fct$ = "$f(x) = "+format$(a,"fix 2")+ &
       "\sin("+format$(b,"fix 2")+"x)+"+ &
       format$(c,"fix 2")+"x^2+"+ &
       format$(d,"fix 2")+"$"

begin key
   pos br nobox
   text fct$
   text "$r^2$ = "+format$(r,"fix 3")
end key
```

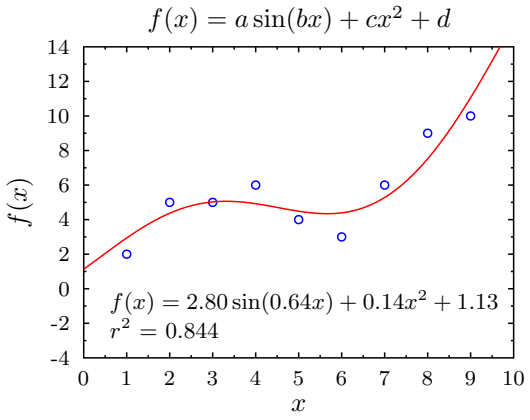


Figure 4.9: Fitting arbitrary curves 'let d2 = fit d1 with ...'.

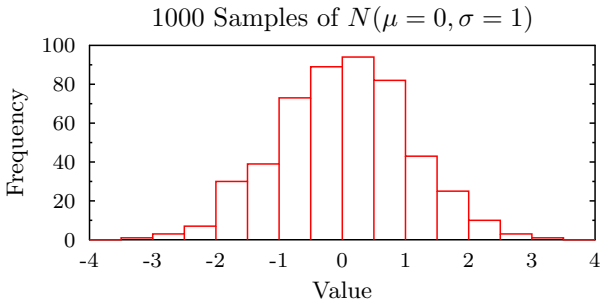


Figure 4.10: An example of 'let dn = hist dn'.

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 scale). If no size command is given, then the size of the graph is initialized to the size of the figure (pagewidth() by pageheight()).

scale h v
: Sets the length of the xaxis to h times the width of the graph box, and the length of the yaxis to v times the height of the graph box. For example, with 'size 10 10' and 'scale 0.7 0.7', the length of the x and y axis will be 7 centimeter. 'scale 1 1' makes the xaxis (yaxis) the same length as the width (height) of the graph box, which is useful for positioning some graphs (see 'fullscale'). The default value for h and v is 0.7.

  If h or v is set to the keyword auto, then the graph is scaled automatically in that direction to fill the entire box. The command 'scale auto' automatically scales the graph in both directions. Note that autoscale also moves the graph, similar to the command 'center'.

  This size command is equivalent to the two commands 'hscale h' and 'vscale v' and allows one to specify the two scale factors with one command.

  Fig. 4.12 shows examples of the different axis scaling options: default, automatic, and 'fullsize'.

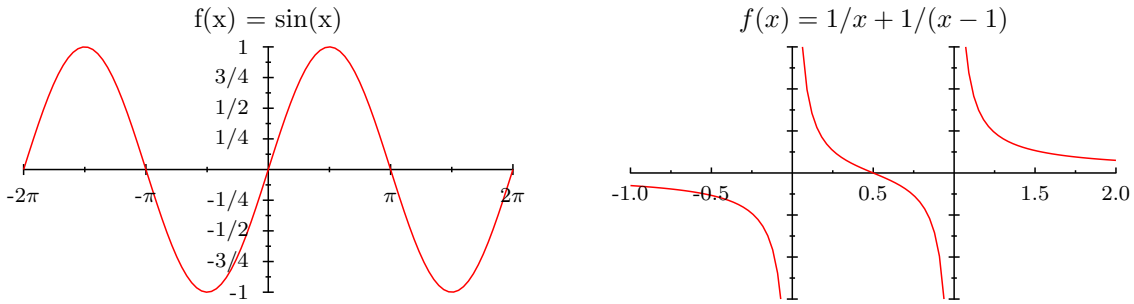


Figure 4.11: Left: math mode graph. Right: graph illustrating the 'origin' option of 'xaxis'.

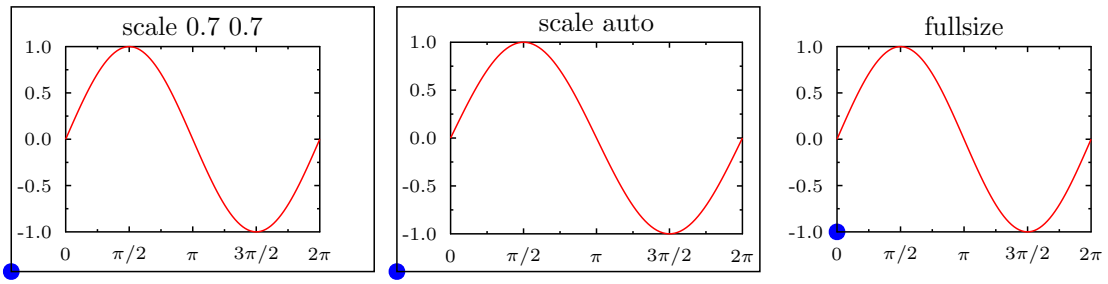


Figure 4.12: Different axis scaling options: default, automatic, and 'fullsize'. The blue dot indicates the origin (x, y) of the graph, that is, the graph is generated with 'amove x y' followed by 'begin graph' ... 'end graph'. The xaxis is labeled using the option 'format pi'.

title *"title"* [hei *ch-hei*] [color *col*] [font *font*] [dist *cm*]
: This command gives the graph a centered title. The list of optional keywords specifies features of it. The dist command is used for moving the title up or down. The default title font size is the value of the '`hei`' setting multiplied by the setting '`titlescale`' (default 1.16).

vscale v
: Sets the length of the yaxis to v times the size of the graph box (default is 0.7). v can also be set to 'auto'. See scale for more details.

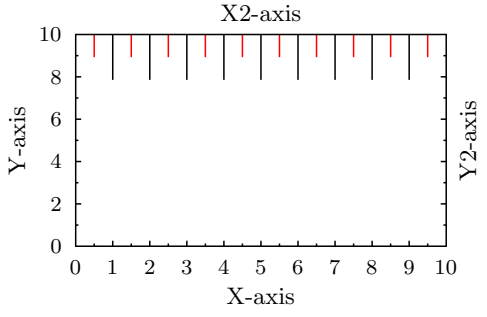
x2labels on
: This command 'activates' the numbering of the x2axis. There is a corresponding command 'y2axis on' which will activate y2axis numbering.

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  "X-axis"
   ytitle  "Y-axis"
   x2title "X2-axis"
   y2title "Y2-axis"
   x2ticks length 0.6
   x2subticks color red
end graph
```



xaxis angle α
: Rotate the labels by α degrees. Fig. 4.17 gives an example.

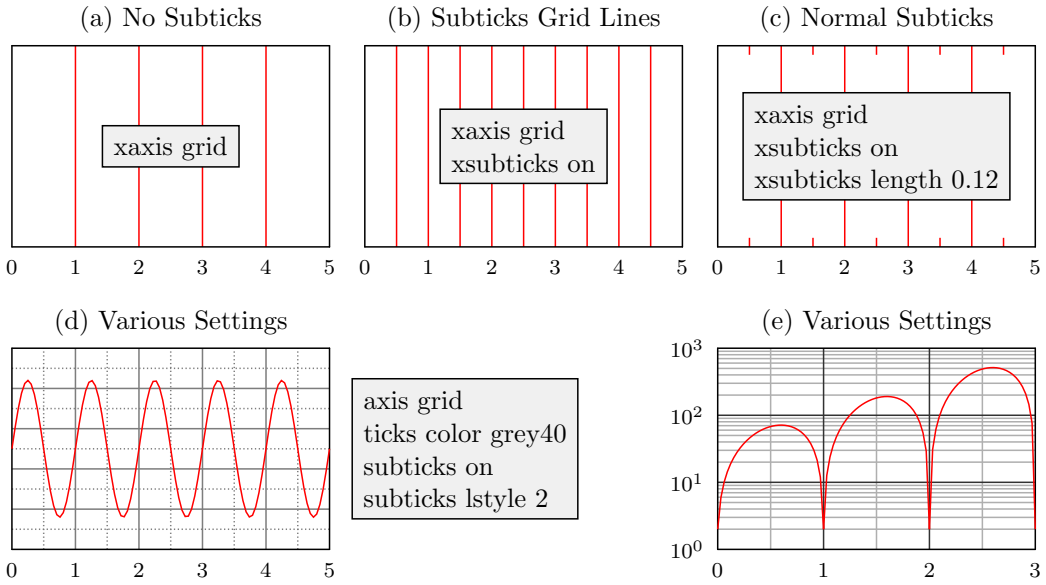


Figure 4.13: Different grid options: no subticks, grid lines at each subtick, or grid lines at main ticks with regular subticks. The box on each graph indicates which GLE commands to use for each option.

xaxis base *exp-cm*
Scale the axis font and ticks by *exp-cm*. The default value of the 'base' setting is the value of 'hei'.

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 52 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.

It is possible to have grid lines for subticks or to have normal subticks. Figure 4.13 shows the different options.

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 labeling 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 labeling 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 3 units 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 ftick *x0* dticks *distance* nticks *number*
: Labels the xaxis starting from position '*x0*', with distance '*distance*' between the ticks. This will result in a tick at the positions $x0 + i \times distance$, with i ranging from 0 to $(number - 1)$.

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

xaxis symticks
: By default, the axis ticks are drawn on the inside of the graph frame. To make them appear on the outside, use a negative ticks length. E.g., 'xticks length -0.1' will produce 1mm ticks on the outside of the graph frame. The 'symticks' option enables tick on both the inside and outside of the graph frame. This option is the default in 'math' mode. (See the 'math' command.)

xlabels [font *font-name*] [hei *char-hei*] [color *col*] [dist *dis*] [on] [off] [log *lgmode*]
: This command controls the appearance of the axis labels. The default label font size is the value of the '`hei`' setting multiplied by the setting '`alabelscale`' (default 0.8). The default value for *dist* is controlled by the setting '`alabeldist`'.

  The command '`xlabels off`' turns the labels for the xaxis off. Similarly, '`xlabels on`' turns them on (the default for the x and y axis, but not for the $x2$ and $y2$ axis).

  Possible values for *lgmode* are: '`off`', '`l1`', '`l25`', and '`l25b`'. These control the subticks for a log scale axis. The value '`off`' means no subticks (i.e., only main ticks at 10^k), '`l1`' means 10 subticks, and '`l25`' means two subticks at the positions $2 \cdot 10^k$ and $5 \cdot 10^k$. The value '`l25b`' is identical to '`l25`' except that now the format function (given with the '`format`' option of the '`xaxis`' command) is used to label the subticks. In the other case, the subticks are labeled with the values '2' and '5' in a small font (0.7 times the size of the main tick labels). These settings are illustrated in Fig. 4.14.

xnames *"name" "name" ...*
: This command replaces the numeric labeling with text labels. Given data consisting of seven measurements, taken from Monday to Sunday, one per day then

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

  would give the desired result (Fig. 4.16). 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.

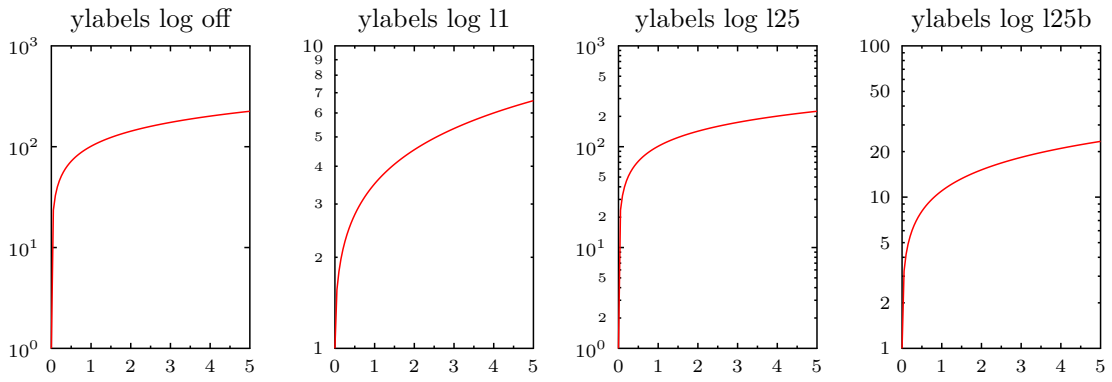


Figure 4.14: Possibilities for the '`log`' option of '`ylabels`'.

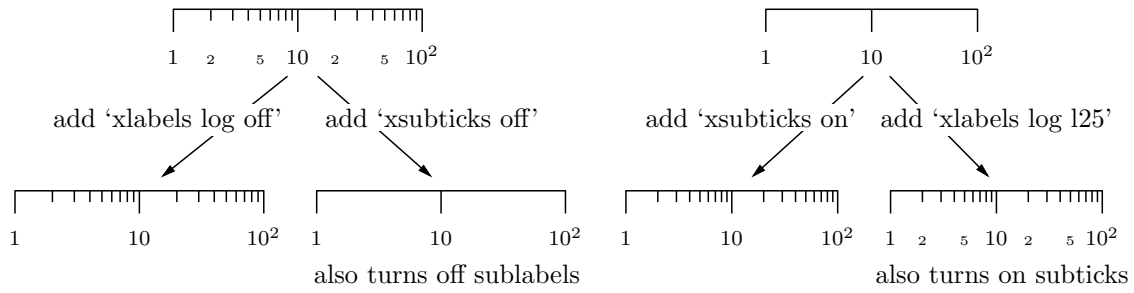


Figure 4.15: How log subticks and log sublabels interact.

xnames from *dx*
: Takes the labels for the xaxis from dataset *dx*. For example, if the data file contains:

```
0, Mercury, 0.382
1, Venus,   0.949
2, Mars,    0.532
3, Jupiter, 11.21
4, Saturn,  9.449
5, Uranus,  4.007
6, Neptune, 3.883
```

then '`ynames from d1`' uses the data from the second column as labels for the yaxis (Fig. 4.17).

xnoticks *pos1 pos2 pos3 ...*
: Disables the axis ticks at the given positions.

xplaces *pos1 pos2 pos3 ...*
: This is similar to the xnames command but it specifies a list of points which should be labeled. This allows labeling which isn't equally spaced. For example:

```
begin graph
   ytitle "Happyness"
   title  "Names \& Places"
   xnames "Mon" "Tue" "Wed" "Thu"
   xnames "Fri" "Sat" "Sun"
   xaxis  min 0 max 6 dticks 1
   ...
end graph
```

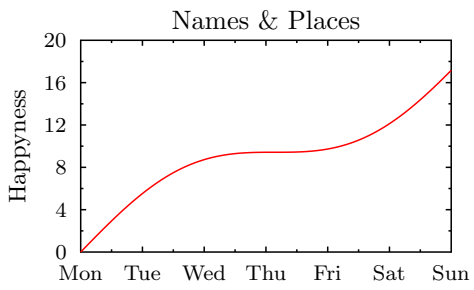


Figure 4.16: Example illustrating the "xnames" command.

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

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

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* on off
: This command gives fine control of the appearance of the axis subticks.

GLE uses an algorithm that decides based on the distance between the main ticks if it should draw subticks or not. However, you can override the decision of this algorithm and explicitly turn the subticks on or off by means of the commands "xsubticks on" or "xsubticks off".

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:

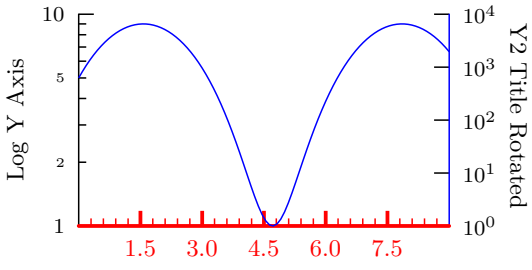
```
xticks length -.2
yticks length -.2
```

xtitle *"title"* [hei *ch-hei*] [color *col*] [font *font*] [dist *cm*] [adist *cm*]
: This command gives the axis a centered title. The list of optional keywords specify features of it. The dist option is used for controlling the distance between the title and the axis labels. The default font size is the value of the '`hei`' setting multiplied by the setting '`atitlescale`' (default 1.0). The default value for *dist* is controlled by the setting '`atitledist`'. The 'adist' option is an alternative to 'dist', but specifies the distance between the title and axis itself. This option is useful for perfectly aligning, e.g., the y-axis titles of multiple graphs (if the graphs are in one vertical column, but their y-axis labels have a different width).

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 nolast
   xaxis hei 0.4 nticks 6 dsubticks 0.3
   xaxis lwidth 0.05 color red
   xticks length 0.2
   ytitle "Log Yaxis"
   yaxis log min 1 max 10
   yticks length 0.2
   y2axis min 1 max 10000 format "sci 0 10"
   y2side color blue
   y2title "Y2title rotated " hei 0.3 rotate
   x2axis off
   y2labels on
   let d1 = sin(x)*4+5 from 0 to 9
   dn line color blue
end graph
```



4.3 Bar Graphs

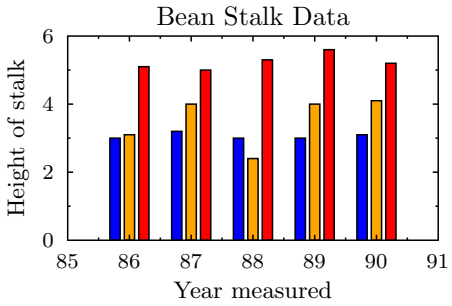
Drawing a bar graph is a subcommand of the normal graph module. This allows bar and line graphs to be mixed. The bar command is quite complex as it allows a great deal of flexibility. The same command allows stacked, overlapping and grouped bars.
For stacked bars use separate bar commands as in the first example below:

```
bar d1 fill black
bar d2 from d1 fill gray10
```

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

```
bar d1,d2,d3 fill gray10,gray40,black
```

```
begin graph
   title  "Bean stalk data" dist 0.1
   xtitle "Year measured"
   ytitle "Height of stalk"
   xaxis dticks 1
   yaxis min 0 max 6 dticks 2
   data  "gc_bean.dat"
   bar d1,d2,d3 fill blue,orange,red
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,...*

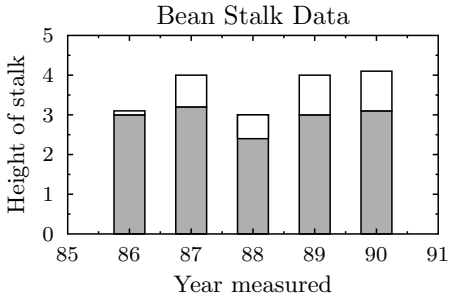
This sets the starting point of each bar in datasets dx,... to be at the value in datasets dy,..., and is used for creating stacked bar charts. Each layer of the bar chart is created with an additional bar command.

```
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
   ...
   data "gc_bean.dat"
   bar d1 fill gray20
   bar d2 from d1 fill white
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 gray10,gray20 color red,green
```

bar *dn,...* fill *f* pattern *p*

The 'pattern' option specifies the pattern used for filling the bars. Fig. 3.7 gives an overview of the predefined patterns that can be used here. Fig. 4.17 shows an example of the command 'bar d2 horiz fill red pattern shade2'.

bar *dn,...* horiz

The option 'horiz' makes the bars horizontal instead of vertical (Fig. 4.17).

4.4 3D Bar Graphs

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

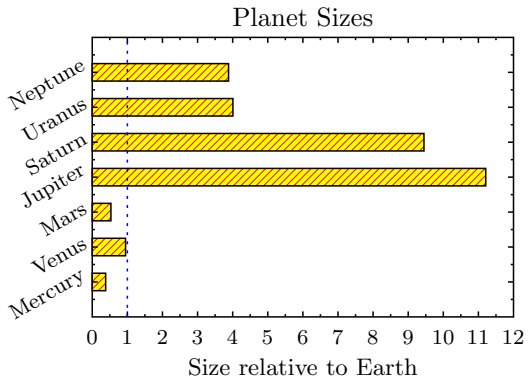


Figure 4.17: Illustration of the 'horiz' and 'pattern' keywords of the 'bar' command, and of the 'angle' option of the 'yaxis' command.

```
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]

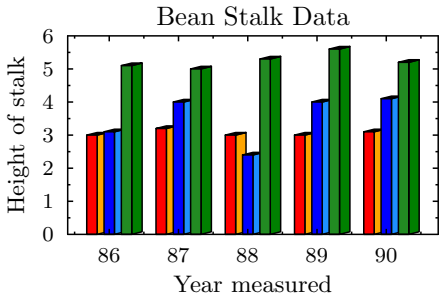
3d *xoffset yoffset*
: Specifies the x and y vector used to draw the receding lines, they are defined as fractions of the width of the bar. A negative xoffset will draw the 3d bar on the left side of the bar instead of the right hand side.

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
   ...
   data "gc_bean.dat"
   bar d1,d2,d3 dist 0.25 width 0.15 3d 1 0.25 &
       fill red,blue,forestgreen &
       side orange,dodgerblue,green
end graph
```



4.5 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.

```
begin graph
   ...
   begin layer 300
      fill x1,d1 color rgba255(255,0,0,80)
      d1 line color red key "$1.5\sin(x)+1.5$"
   end layer
   begin layer 301
      fill x1,d2 color rgba255(0,128,0,80)
      d2 line color green key "$1/x$"
   end layer
   ...
end graph
```

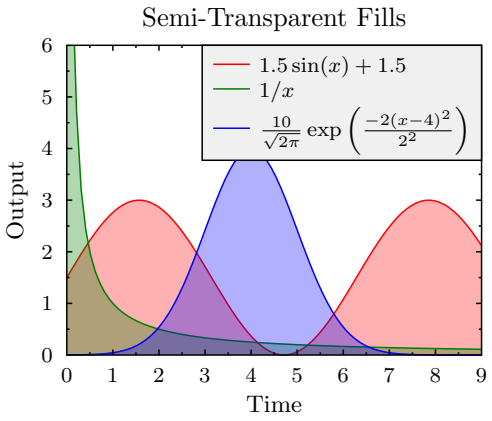


Figure 4.18: A graph with semi-transparent fills.

```
begin graph
   title "$r = \cos(2\theta)$"
   xaxis min -1.1 max 1.1
   yaxis min -1.1 max 1.1
   begin layer 0
      draw polar_grid radius 2 rings 5 sectors 12
      draw polar "cos(2*t)" from 0 to 2*pi fill wheat
   end layer
end graph
```

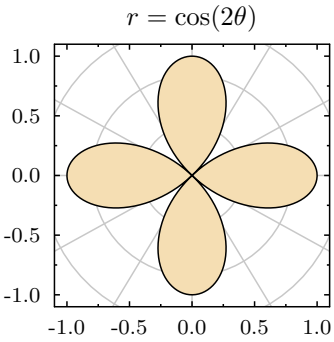
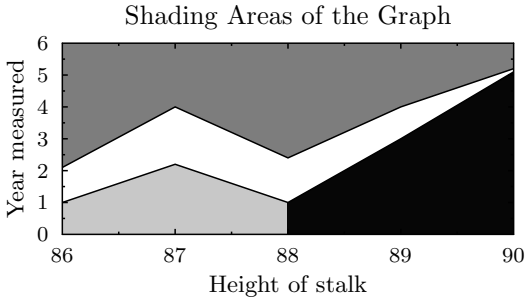


Figure 4.19: Polar plots using the "draw" command and the "polar" function from "polarplot.gle".

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.

```
begin graph
   title  "Shading areas of the graph" dist 0.1
   xtitle "Height of stalk"
   ytitle "Year measured"
   xaxis min 86 max 90
   yaxis min 0  max 6
   data "gc_fill.dat"
   fill d2,x2 color gray40
   fill x1,d1 color gray10 xmin 85 xmax 88
   fill x1,d1 color gray90 xmin 88 xmax 91
   dn line
end graph
```



To create semi-transparent fills, use the function "rgba255" to specify the fill color. This function allows one to define a semi-transparent color with red, green, blue, and alpha components. The alpha component defines the transparency. Fig. 4.18 shows an example. To create output with semi-transparent colors, GLE's command line option "-cairo" must be used.

4.5.1 Polar Plots

GLE supports polar plots by means of the "polar" and "polar_grid" functions from the library file "polarplot.gle". See Fig 4.19 for an example. This example also illustrates the use of the "draw" function (p. 29) and the "layer" block (p. 42). The "layer" block is used here to make sure that the polar grid is drawn before the axis are drawn.

4.6 Notes on Drawing Graphs

4.6.1 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
```

4.6.2 Layers

GLE draws a graph as a sequence of layers. The following are the default layers.

200 A grid (e.g., "yaxis grid")

350 Fill type graphs (p. 40)

350 Bar type graphs (p. 38)

500 Graph axis

700 Line type graphs

700 Error bars

700 Marker type graphs

700 Draw commands (p. 29)

There are only 4 default layers. Each layer has a unique number (200, 350, 500, and 700). Layers are drawn from small to large. Within a layer, the elements are drawn in the order indicated above. For example, error bars are drawn after line type graphs.
It is possible to define new layers with the "begin layer / end layer" block. This is illustrated by the following example.

```
begin graph
   data "file.csv"
   ...
   begin layer 400
      d1 line color red
   end layer
end graph
```

This example defines a custom layer with the unique number 400 and one line type graph. The result will be that d1 is drawn after any defined bar type graphs and before the graph x/y axis.
More examples of layers can be found in the following figures:

- Fig. 4.7 shows how a layer can be used to draw a custom graph background.
- Fig. 4.18 shows how layers can be used to combine fill type and line graphs.
- Fig. 4.19 shows how layers can be used to draw a custom grid.

4.6.3 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.

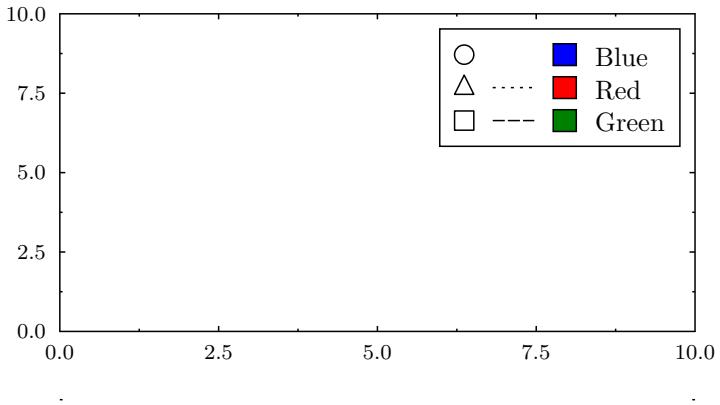
```
size 10 10
set lwidth .1
begin graph
   ...
end graph
```

Chapter 5

The Key Module

The key module is used for drawing keys. The key can be either specified through a separate key block or directly in the graph block (by prefixing the key commands with the keyword "key"). This chapter first discusses how to define the key using a key block. Section 5.3 shows how to include the key commands directly in a graph block.
The key block usually comes directly after the graph block as follows:

```
begin graph
   ...
end graph

begin key
   position tr
   offset 0.2 0.2
   text "Blue"    marker circle   fill blue
   text "Red"     marker triangle fill red    lstyle 2
   text "Orange"  marker square   fill orange lstyle 3
end key
```



The key block consists of two parts: (a) global commands, and (b) the definitions of the entries. Global commands appear at the beginning of the key and define, e.g., the position of the key. In the example, "position" and "offset" are global commands. Multiple global commands are allowed on a given line. The entry definitions start after the global commands. All commands relevant to a given entry must appear on the same line. In the example, there are three entry definitions and each definition starts with the "text" command. Entries can be organized into columns using the "separator" command.
There are two possible ways to set the position of a key: (a) the key can be positioned relative to the graph, and (b) it can be positioned at given coordinates. To position the key relative to the graph, use the commands "position" and (optionally) "offset". For example,

```
position tr
offset 0.2 0.2
```

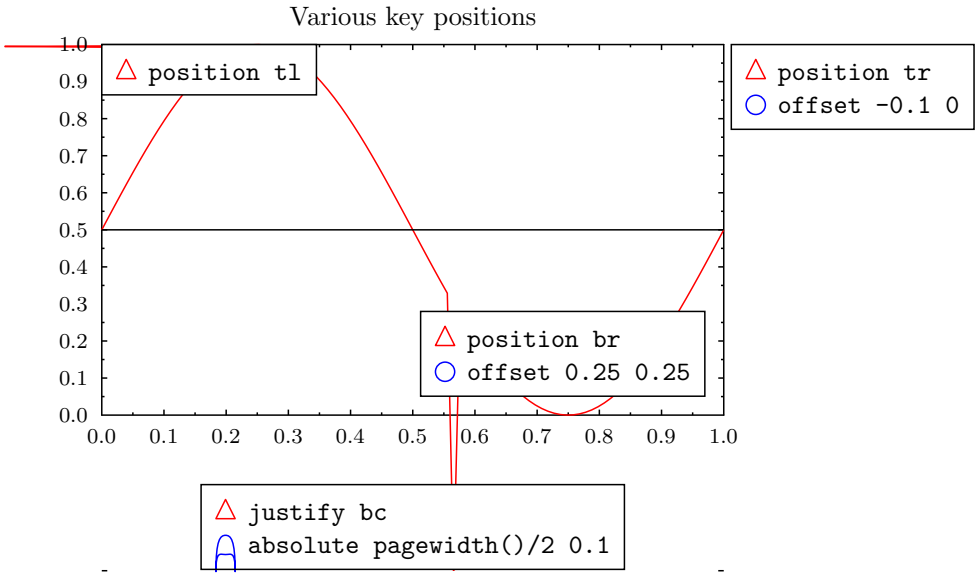


Figure 5.1: Various positions for the key.

places the key at the top-right corner of the graph 0.2 cm from each side. To position the key at given coordinates use the "justify" and "absolute" commands. For example,

```
justify bc
absolute 5 0.1
```

places the bottom-center of the key at position (5 cm, 0.1 cm). Fig. 5.1 gives some examples of positioning the key.

5.1 Global Commands

Global commands appear at the start of the key block. They control the position of the key and various other properties of the key. Several global key commands may appear on one line in the script.

absolute x y
: Places the key at position (x, y) on the figure. The anchor point of the key is specified with the "justify" command.

base h or row h
: Sets the base scale of the entries. The sizes of all components are initialized based on this. E.g., to change the size of the filled box in an entry, use this command.

boxcolor c
: Set the background color of the key to c.

coldist d
: Sets the distance between columns. (To obtain a key with multiple columns, use the "separator" command.)

compact
: Creates a more compact key by combining the "line" and "marker" fields into one field. The effect of this is shown in Fig. 5.3.

dist d
: Sets the distance between the different components of an entry (the marker, the line, the fill, and the text).

hei h
: Sets the height of the text in the entries of the key. If this command is not given, then the current height is used. (To set the current height, use "set hei", see page 20.)

justify x
: Sets the anchor point of the key. Possible values: tl, bl, tr, br, tc, bc, lc, rc, cc. These stand for top-left, bottom-left, top-right, bottom-right, top-center, bottom-center, left-center, right-center, and center. Use this command in combination with the "absolute" command. Fig. 5.1 gives some examples.

llen x
: Sets the length of the line in the entries.

lpos x
: Sets the vertical position of the line in the entries. (This is normally set automatically.)

margins x y
: Sets the margins of the key block. (The space between the border and the entries.)

nobox
: Do not draw a border around the key.

off
: Disable this key.

offset x y
: Specifies the distance in cm between the position specified with the "position" or "pos" command and the actual key. A negative offset places the key outside of the graph (Fig. 5.1).

position x or pos x
: Specifies the position of the key on the graph. Possible values: tl, bl, tr, br, tc, bc, lc, rc, cc. These stand for top-left, bottom-left, top-right, bottom-right, top-center, bottom-center, left-center, right-center, and center. Optionally, the "offset" command can be combined with this command. Fig. 5.1 gives some examples.

5.2 Entry Definition Commands

Each entry in the key is represented by one line in the key block, and all commands for a given entry must appear on that line. The following commands can be used to define key entries.

color c
: Sets the color of the line and marker. The other components of the key are drawn in the default color. (To set the default color, use "set color", see page 19.)

fill p
: Sets the fill color or pattern.

line
: Shorthand for "lstyle 1".

lstyle s
: Sets the line style.

lwidth
: Sets the width of the line.

marker m
: Sets the marker.

mscale x
: Sets the scale of the marker.

msize x
: Sets the size of the marker.

pattern x
: Sets the filling pattern. Fig. 3.7 shows examples of filling patterns.

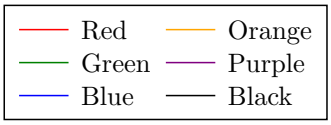


Figure 5.2: Defining a key with multiple columns.

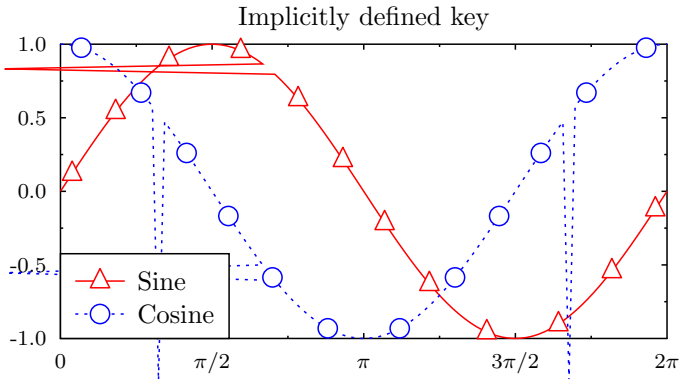


Figure 5.3: Defining the key together with the graph block. This figure also illustrates the 'mdist' option of the 'marker' command.

separator [lstyle *s*]
: Use this command to divide the key into multiple columns. If the "lstyle" option is given, then a line is drawn between the columns in the given style. Possible values are given with the description of the "set lstyle" command on page 21. The "separator" command should be inserted between the key entries that should go in different columns. For example,

```
begin key
   position bl
   line color red    text "Red"
   line color green  text "Green"
   line color blue   text "Blue"
   separator
   line color orange text "Orange"
   line color purple text "Purple"
   line color black  text "Black"
end key
```

will result in the key shown in Fig. 5.2.

text *s*
: The text for the entry.

5.3 Defining the Key in the Graph Block

It is also possible to define the key in the graph block itself. This is accomplished by prefixing global key commands with the keyword "key". The entries are in this case defined with the "dn" commands and the labels are set with the "key" option to these commands.
The following presents an example:

```
begin graph
   title "Implicitly defined key"
   let d1 = sin(x)
   let d2 = cos(x)
   xaxis min 0 max 2*pi dticks pi/2 format "pi"
   key compact pos bl
```

```
   d1 line color red  marker triangle mdist 1 key "Sine"
   d2 line color blue marker circle   mdist 1 lstyle 2 key "Cosine"
end graph
```

Fig. 5.3 shows the result.
It is also possible to put a "key separator" line in between the "dn" lines create a key with multiple columns. For example:

```
   d1 line color red key "Sine"
   key separator
   d2 line color blue key "Cosine"
```

If you plot data from a data file, and the first row of the file contains column labels, then these labels will be used automatically to construct the key. However, if you prefer to construct the key manually instead by defining a key block, then you can override this behavior by making GLE ignore the row with the labels using the "ignore" option of the "data" command. E.g., `data "myfile.csv" ignore 1` (see p. 25). Alternatively, you can accomplish the same by adding the command "key off" to the graph block to disable the automatically generated key.

Chapter 6

Programming Facilities

6.1 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.

6.2 Functions Inside Expressions

eval(str)
: Evaluates the given string as if it was a GLE expression and returns the result. E.g., `eval("3+4")` returns 7.

arg(i), arg$(i), nargs()
: Provide access to the command line arguments that are passed to GLE. This is useful for generating multiple similar plots from a single script. arg(i) returns the i-the argument (as a number), arg$(i) returns the i-the argument as a string, and nargs() returns the number of arguments. Only arguments that come after the name of the GLE script are counted. For example, if GLE is run as:

  ```
  gle -o graph-1.eps graph.gle "Title" 0.5
  ```

  then nargs() returns 2, arg$(1) returns "Title", and arg(2) returns 0.5.

  The typical use of these functions is to create a script "graph.gle" as follows:

  ```
  size 10 10
  begin graph
     title arg$(1)
     data arg$(2)
     d1 line color red
  end graph
  ```

  and then creating different graphs by running GLE multiple times:

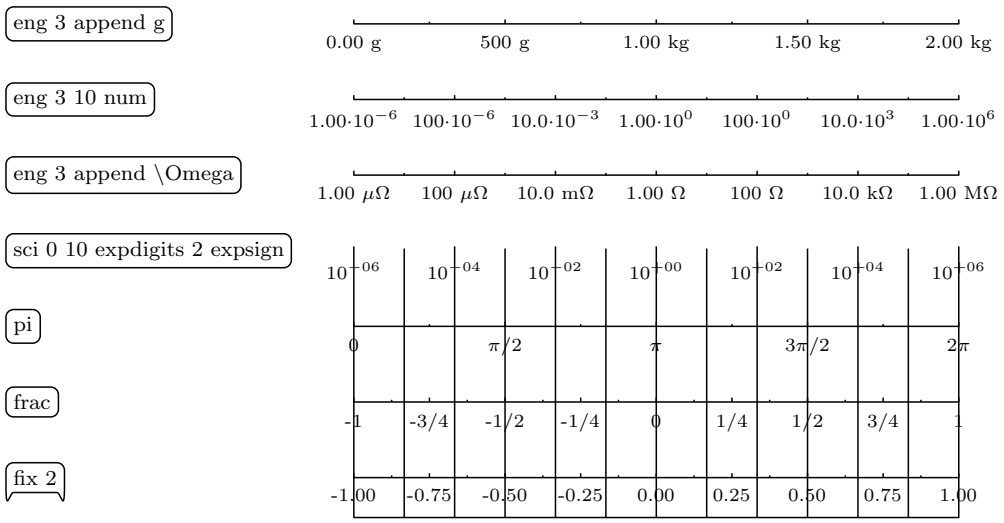


Figure 6.1: Examples of number formatting options.

```
gle -o beans.eps graph.gle "Beans" "beans.csv"
gle -o peas.eps graph.gle "Peas" "peas.csv"
```

This will create two graphs: "beans.eps" and "peas.eps". The arg() functions can be used at all places in the script where an expression is expected. They can even be used in place of GLE commands in a graph block by means of the \expr() function. For example,

```
data "file.csv"
d\expr{arg(1)} line color red
```

in the graph block will allow one to draw different datasets from a single file on multiple graphs. To do so, run:

```
gle -o d1.eps graph.gle 1
gle -o d2.eps graph.gle 2
```

dataxvalue(*ds,i*), datayvalue(*ds,i*), ndata(*ds*)

These functions can be used to retrieve the data values from a given dataset. They only work after data has been loaded by means of a graph block (Ch. 4).

A dataset is specified with a dataset identifier *ds* (a string of the form "d*i*", with *i* an integer). The function 'ndata' returns the number of points in the dataset, and the functions 'dataxvalue' and 'datayvalue' return the x and y values of point i, where i ranges from 1 to ndata(*ds*).

The following example shows how these functions can be used to compute the maximum of a dataset:

```
sub dmaxy ds$
   local crmax = datayvalue(ds$,1)
   for i = 2 to ndata(ds$)
      crmax = max(crmax, datayvalue(ds$,i))
   next i
   return crmax
end sub
```

This subroutine together with other subroutines for computing the minimum, mean, area, etc. of a dataset are defined in the include file 'graphutil.gle', which is distributed together with GLE.

format$(*exp,format*)

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

Basic formats:

- append *s*: appends the string *s* after the formatted number. This can be used to add a unit.

- 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.
- eng *digits* [e,E,10] [expdigits *num*] [expsign] [num]: format in engineering notation. The options are similar to 'sci'. If the option 'num' is given, then numeric notation is used instead of, e.g., μ, m, k, M.
- round *sig*: format a number with *sig* significant digits.
- frac: format the number as a fraction.
- pi: format the number as a fraction times π (E.g., xaxis labels of Fig. 4.12).

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.
- prepend *s*: prepends the string *s* before the formatted number.
- 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") = $3.8 \cdot 10^{03}$

Several formats can be combined into one string: "sci 2 10 min 1e2 fix 0" uses scientific notations for numbers above 10^2 and decimal notation for smaller numbers. See Fig. 6.1 for more examples.

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(str), theight(str), tdepth(str)
: These functions return the width, depth and height of a string, if it was printed in the current font and size.

width(obj), height(obj)
: 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 just left
text abc
set color gray20
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 gray10
```

xpos(), ypos()
: Returns the current x and y points.

See Appendix A.2 for an overview of all functions provided by GLE.

6.3 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 store numbers like 1234.234.

```
name$ = "Joe"
height = 6.5   ! Height of person
shoe = 0.05    ! shoe adds to height of person
amove 1 1
box 0.2 height+shoe
write name$
```

6.4 String constants

String constants can be double quoted or single quoted. To include a double quote character in a double quoted string it should be doubled. The same holds for a single quoted string. Backslash characters are not interpreted in a special way by GLE's parser. (They are, however, interpreted by the built-in TEX system used in the 'write' command.) Here are some examples:

```
print "Double quoted string"
print 'Single quoted string'
print "Between ""double quotes"""
print "{\it hello}"
print "Three double quotes """""""
print """"
```

The result of these print commands is:

```
Double quoted string
Single quoted string
Between "double quotes"
{\it hello}
Three double quotes """
"
```

6.5 Programming Loops

The simple way to draw a 6 × 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
```

For-next loops, and all other control constructs, can also be used among others inside graph and key blocks. This is useful for drawing many similar functions (Fig. 4.6). Besides for-next loops, GLE also supports while and until loops:

```
i = 0
while i <= 10
   print "Value: " i
   i = i + 1
next


i = 0
until i > 10
   print "Value: " i
   i = i + 1
next
```

6.6 If-then-else

GLE supports if-then-else statements as follows:

```
if a < 1 then print a "is smaller than 1"
else if a < 2 then print a "is smaller than 2 but larger than 1"
else if a < 3 then print a "is smaller than 3 but larger than 2"
else print a "is larger than 3"
```

to create blocks of code for the ‘then’ and ‘else’ branches, instead use:

```
if a < 1 then
   print a "is smaller than 1"
   ...
else
   ...
end if
```

More complex conditions can be created with the logic connectives ‘and’, ‘or’, and ‘not’ (note the parenthesis around the logical expressions):

```
if (a >= 1) and (a <= 10) then print "a is between 1 and 10"
```

6.7 Subroutines

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

```
sub grid nx ny
   local x, y
   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

amove 2 4
grid  6 8
amove 2 2
grid  9 5
```

Inside a subroutine, the keyword '`local`' can be used to define local variables. E.g., '`local x = 3`', defines the local variable 'x' and assigns it the value 3. It is also possible to define several local variables at once, as is shown in the 'grid' example above.
The keyword '`return`' can be used to return a value from a subroutine. E.g.,

```
sub gaussian x mu sigma
   return 1/(sigma*sqrt(2*pi))*exp(-((x-mu)^2)/(2*sigma^2))
end sub
```

The main GLE file will be much easier to manage if subroutine definitions are moved into a separate file:

```
include "griddef.gle"

amove 2 4
grid  2 4
amove 2 2
grid  9 5
```

More information about the "include" command can be found on page 14.

6.7.1 Default Arguments

Given the following subroutine definition:

```
sub mysub x y color$ fill$
   default color  "black"
   default fill   "clear"
   print "Color: " color$
   print "Fill:  " fill$
end sub
```

the following calls are valid:

```
mysub 1 0
mysub 1 0 red
mysub 1 0 red green
mysub 1 0 fill blue
mysub 1 0 color red
mysub 1 0 color red fill blue
```

6.8 Forward Declarations

A forward declaration of a subroutine is possible with:

```
declare sub mysub x y
```

Forward declarations are required for declaring mutually recursive subroutines.

6.9 I/O Functions

The following I/O functions are available:

fopen *name file* [read|write]
: Open the file "*name*" for reading or for writing. The resulting file handle is stored in variable "*file*" and must be passed to all other I/O functions.

fclose *file*
: Close the given file.

fread *file x1 ...*

freadln *file x1 ...*
: Read entries from "*file*" into given arguments *x1 ...*

fwrite *file x1 ...*
: Write given arguments to "*file*".

fwriteln *file x1 ...*
: Write given arguments to "*file*" and start a new line.

fgetline *file line$*
: Read an entire line from "*file*" and store it in the string "*line$*".

ftokenizer *file commentchar spacetokens singletokens*
: Sets up the parameters of the tokenizer that controls the reading of "*file*". The *commentchar* parameter specifies the characters that are to be interpreted as line comments. It is a string, but each character of the string is assumed to be a separate comment character. The default is "!". If one would write "!%", then both "!" and "%" would be comment indicators. The fread functions skip everything after a comment character to the end of the line. The *spacetokens* string represents the set of characters that are interpreted as spaces or delimiters. The default value is " ,\t\r\n", i.e., space, comma, tab, carriage return, and newline are delimiters by default. Finally, the *singletokens* string identifies characters that should be returned as separate tokens, even if they are glued to other tokens. For example, if "@" would be a single char token, then the string "me@myself.com" would be returned in three tokens: "me", "@", and "myself.com".

For example:

```
fopen "file.dat" f1 read
fopen "file.out" f2 write
until feof(f1)
        fread f1 x y z
        aline x y
        rline x z
        fwriteln f2 x*2 "y =" y
next
fclose f1
fclose f2
```

6.10 Device Dependent 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 7

Advanced Features

This chapter covers the advanced features of GLE.

7.1 Diagrams

7.1.1 Named Boxes and the Join Command

GLE can name objects using the "begin/end name" (p. 10), "begin/end box" (p. 9), and "begin/end object" (p. 10) constructs, and using the "name" option supported by some drawing commands. The name is always associated with the rectangular region on the figure that corresponds to the bounding box of the object (the smallest rectangle that surrounds all points of the object). The following example shows how to create a blue rectangle named "square" and a box with the text "Title" named "titlebox".

```
amove 1 1
box 1 1 fill blue name square

amove 5 5
begin box add 0.1 name titlebox
   write "Title"
end box
```

The "join" command (p. 15) can now be used to draw lines or curves (optinally with arrows) between designated points of the named objects. The following example shows how to draw an arrow from the top-right (".tr") point of the blue square to the bottom-centre point (".bc") of the object named "titlebox".

```
join square.tr -> titlebox.bc
```

The "`->`" in the join command indicates that the arrow should go from the first object towards the second. The symbol "`<-`" is used to draw the arrow in the opposite direction. A bidirectional arrow is obtained with "`<->`" and a line without an arrow is obtained with "`-`". The join command can also create Bezier curves instead of straight lines. See the command's description on p. 15 for more information.

```
join square.tr ->  titlebox.bc
join square.tr <-  titlebox.bc
join square.tr <-> titlebox.bc
join square.tr -   titlebox.bc
```

The named points (corners, centre points, ...) on each named object are indicated as defined in Table 7.1. ".box" clipping is the default and can be omitted.
It is also possible to name an individual point (instead of an object). To do so, simply move there and save that point as a named object.

```
amove 2 3; save apoint
join apoint - square
```

Table 7.1: Justify options for the join command.

.tr	Top right
.tc	Top centre
.tl	Top left
.bl	Bottom left
.bc	Bottom centre
.br	Bottom right
.rc	Right centre
.lc	Left centre
.cc	Centre
.v	Vertical line
.h	Horizontal line
.c	Circle/ellipse clipping (for drawing lines to, e.g., a circle)
.box	Box clipping

The functions "pointx" (abbreviated to "ptx", see p. 53), "pointy" (abbreviated to "pty"), "width", and "height" (p. 53) apply to named points and objects. They retrieve the "x" respectively "y" coordinate of a named point, and the width and height of a named object. For example:

```
print ptx(square.tr)
print pty(square.tr)
print width(square)
print height(square)
```

Complete Example

Below is a complete example that makes use of the constructs described above. The resulting figure is shown in Fig. 7.1.

```
begin name line
   amove 8 18
   rline 0 -6
end name

begin name main
   amove 9.5 6.5
   ellipse 2 0.8
   write "Main"
end name

amove 3 16
begin box name grv add 0.3 round 0.3 fill lightcyan
   write "GRV"
end box

amove 12.5 16.5
begin box name cheese add 0.3 fill lightcyan
   write "Cheese"
end box

amove 15.5 11.5
begin box name chv add 0.3 fill lightcyan
   write "CHV"
end box

amove 3 10
begin box name goats add 0.3 fill lightcyan
   write "Goats"
```

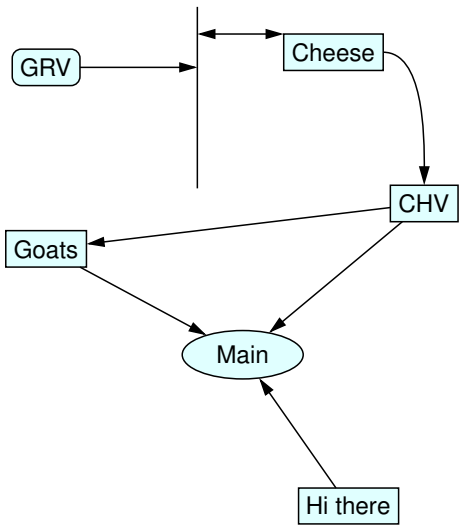


Figure 7.1: Joining named points.

```
end box

amove 13 1.5
begin box name hi add 0.3 fill lightcyan
   write "Hi there"
end box

join chv -> goats                          ! ".box" is default and can be omitted
join grv -> line.h                         ! ".h" means to join horizontally
join line.h <-> cheese.tl
join cheese.rc -> chv.tc curve 0 90 1.5 1
join main.c <- hi                          ! ".c" is used for circles
join main.c <- chv
join main.c <- goats
```

7.1.2 Object Blocks and Hierarchically Named Points

The "begin/end name" (p. 10) construct names the object that results from the drawing commands in this block. The "begin/end object" (p. 10) is similar, but it does not actually draw the object. It rather defines an object that can be drawn later by means of the "draw" command (p. 13). An object block is therefore very similar to a subroutine. It actually works in the same way as a subroutine, but it is 'executed' by the "draw" command rather than by a regular subroutine call. An object block can also have parameters. Here is an example of an object block that defines a house.

```
begin object house
   ! draw a house with a named door and window
   set join round
   ! draw the roof
   begin path stroke fill lightsalmon
      amove 0 1.625
      aline 1.25 2.5
      aline 2.5 1.625
      closepath
   end path
   ! draw the brick wall
   amove 0 0
   box 2.5 1.625 fill cornsilk
```

```
   ! draw the door
   amove 1.5 0
   box 0.75 1.375 fill burlywood name door
   ! draw the window
   amove 0.25 0.625
   box 1 0.75 fill skyblue name window
end object
```

To draw the house defined by the above block, one uses the following draw command. The first argument of the command is the name of the object followed by a dot followed by a justify option (Table 7.1). The justify option is used to position the object. In this case, the house is drawn such that its bottom-centre point is horizontally in the middle of the figure and at a height of 1.5 cm.

```
amove pagewidth()/2 1.5
draw house.bc
```

The "draw" command names the object using the same name as the name of the object block by default. An alternative name can be given using its "name" option. In this example, the resulting object on the figure will be called "house". Note that the object definition for the house also includes names for the sub-objects "door" and "window". These names can be accessed using the "dot" notation as follows:

```
print ptx(house.door.cc)
print pty(house.door.cc)
```

These so-called hierarchical object names can also be used to position the object. The following example draws the house such that its door's centre point is at position (5, 5).

```
amove 5 5
draw house.door.cc
```

By using the "draw" command inside an object's definition, names can be arbitrary nested. For example, if "door" would be defined as an object block that includes the name "handle", and if the object block defining "house" would include a draw command to draw the object "door", then the global name of the door's handle becomes "house.door.handle".
The hierarchical object names can be used to refer to points on the object. The following example shows how these can be used with the pointx and pointy function to draw labels on the figure.

```
set just lc
amove pointx(house.rc)+0.5 pointy(house.door.cc)
begin name doorlabel add 0.05
   write "house.door.cc"
end name

set just rc
amove pointx(house.lc)-0.5 pointy(house.window.cc)
begin name windowlabel add 0.05
   write "house.window.cc"
end name

join windowlabel.rc -> house.window.cc
join doorlabel.lc -> house.door.cc
```

The resulting figure is shown in Fig. 7.2. The file "shape.gle" that is distributed with GLE contains object block definitions for many useful shapes.

7.2 LaTeX Interface

7.2.1 Example

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

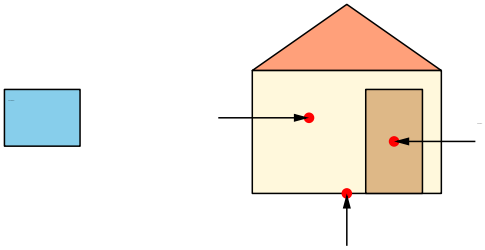
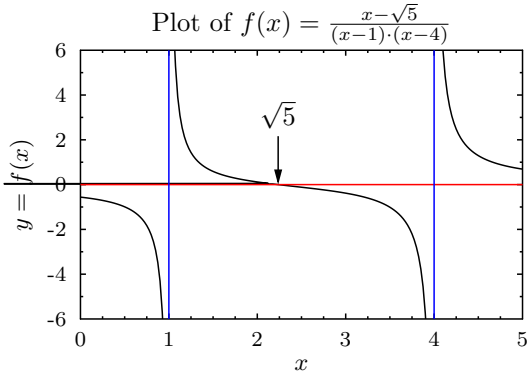


Figure 7.2: Drawing objects and hierarchically named points.

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

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

amove xg(sqrt(5)) yg(0); save sq50
join sq5b.bc -> sq50
```



LaTeX expressions are drawn on top of all other graphics and cannot clipped (Sec. 7.3).
LaTeX expressions respect the '`just`' setting and, depending on the value of '`texscale`', also the '`hei`' setting (the font size). If '`set texscale scale`' is used, then LaTeX expressions are scaled to the value of '`hei`'. If '`set texscale none`' is used, then LaTeX expressions are not scaled. 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 LaTeX (e.g., `\large`, ...). Finally, if '`set texscale fixed`' is used, then the default LaTeX size that most closely matches the value of '`hei`' is selected.

7.2.2 Using LaTeX Packages

If your LaTeX expressions require special LaTeX packages, these can be loaded using the `texpreamble` block. E.g., put the following near the beginning of your GLE file:

```
begin texpreamble
    \documentclass{llncs}
    \usepackage{amsmath}
    \usepackage{amssymb}
    \DeclareMathSymbol{\R}{\mathbin}{AMSb}{"52}
end texpreamble
```

7.2.3 Using UTF-8 Encoding in GLE Scripts with LaTeX Expressions

If you save your .gle files in Unicode (UTF-8) encoding, then you can type accented characters (such as é, ü, ž, ...) directly into your GLE script. In order to also allow such encoded characters in LaTeX expressions, add the following 'texpreamble' to your GLE script:

```
size 4 4
begin texpreamble
   \usepackage[utf8]{inputenc}
end texpreamble
amove 1 2; tex "éüž"
```

7.2.4 Import a GLE Figure in a LaTeX Document

There are two methods for importing the output of a GLE file with TEX expressions in your LATEX document. The most obvious one is by just importing the .eps/.pdf file generated by GLE with \includegraphics. E.g., if you have a GLE script 'sin.gle' and you run 'gle -d eps -d pdf sin.gle' to produce the .eps/.pdf output, then you could include this in a LaTeX document as follows:

```
\documentclass{article}
\usepackage{graphics}
\begin{document}
   \begin{figure}
      \includegraphics{sin}
      \caption{\label{sin}The sine function.}
   \end{figure}
\end{document}
```

An alternative method is by using GLE's command line option '-inc'. If this option is supplied, then GLE will create besides the usual .eps or .pdf file also an .inc file. This .inc file can be imported in a 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 TEX draw commands for drawing the LATEX expressions on top of the GLE output. Note that the .eps/.pdf file created by GLE does not include these if -inc is used (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 LATEX document.

```
\usepackage{graphics}
\usepackage{color}
```

If you place your .gle files in a subdirectory of the directory where your 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/" -inc myfile.gle
```

GLE can color and rotate 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.
The main advantage of using the -inc method is that the resulting file size will be smaller because the LATEX fonts are not included in the .eps/.pdf file generated by GLE.

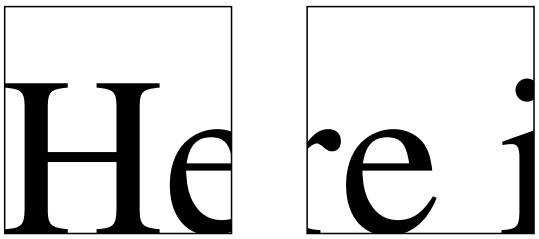
7.2.5 The .gle Directory

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

7.3 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
```



7.4 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. A sample of the predefined color names is included in Appendix A.8.
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 `rgb255` primitive.

```
mycolor$ = "rgb255(38,38,134)"
```

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
```

7.5 GLE's Configuration File

GLE reads two configuration files during initialization. The first configuration file is the file "glerc" located in the root of your GLE installation. This location is usually referred to as $GLE_TOP. To find out where your $GLE_TOP is, run "`gle -info`". The second configuration file is the file ".glerc" located in your home directory (Unix and Mac OS/X only). The commands in this second file override the commands in $GLE_TOP/glerc.
The configuration files can be used to set various options, such as the paper size and margins.
To set the paper size and margins, add the following block to the configuration file.

```
begin config paper
     size = letterpaper
     margins = 2.54 2.54 2.54 2.54
end config
```

The supported paper sizes are listed with the description of the "`papersize`" command on page 16.
The configuration file can also be used to override default locations of external tools such as GhostScript and LaTeX.

```
begin config tools
     ghostscript = $HOME/bin/gs
     pdflatex = /usr/bin/pdflatex
     latex = /usr/bin/latex
     dvips = /usr/bin/dvips
     dvips_options = "-j0"
end config
```

Note that GLE expands environment variables in the tool locations. If I'm john, then GLE will search for GhostScript in `/home/john/bin/gs` in the above example. It is also possible to specify additional command line options to be passed to the tools by means of `ghostscript_options`, `pdflatex_options`, `latex_options`, and `dvips_options`. In the above example, the option "`-j0`" will be passed to dvips. As a result, it will not subset fonts.

Chapter 8

QGLE: GLE's Graphical User Interface

QGLE is the graphical user interface (GUI) to GLE. A screenshot of QGLE with some of its dialogs appears in Fig. 8.1.
The current GUI contains a preview window that can receive messages from GLE and display the resulting EPS file. In addition, it can open GLE and EPS files directly (using Ghostscript and GLE). It has the capability to add and edit simple objects, such as, lines, circles, arcs (snapping to a grid if required), and complex object blocks (defined with begin/end object). It can also alter various properties of the objects, such as, line width, and color. The perpendicular line and tangential line commands can be used to produce a line starting perpendicular or tangential to an existing object. OSnap can be used for the end point of a line.

When drawing objects, hit the escape key to cancel. The pointer tool can be used to select objects and they can be deleted using the 'del' key or moved/scaled. Shift can be used to select multiple objects.

Keyboard shortcuts include:

F1	Open the GLE manual
F3	Toggle object snap
F6	Toggle coordinate display
F7	Toggle grid visibility
F8	Toggle orthographic snap
F9	Toggle grid snap
F10	Toggle polar snap
'-'	Zoom in
'='	Zoom out

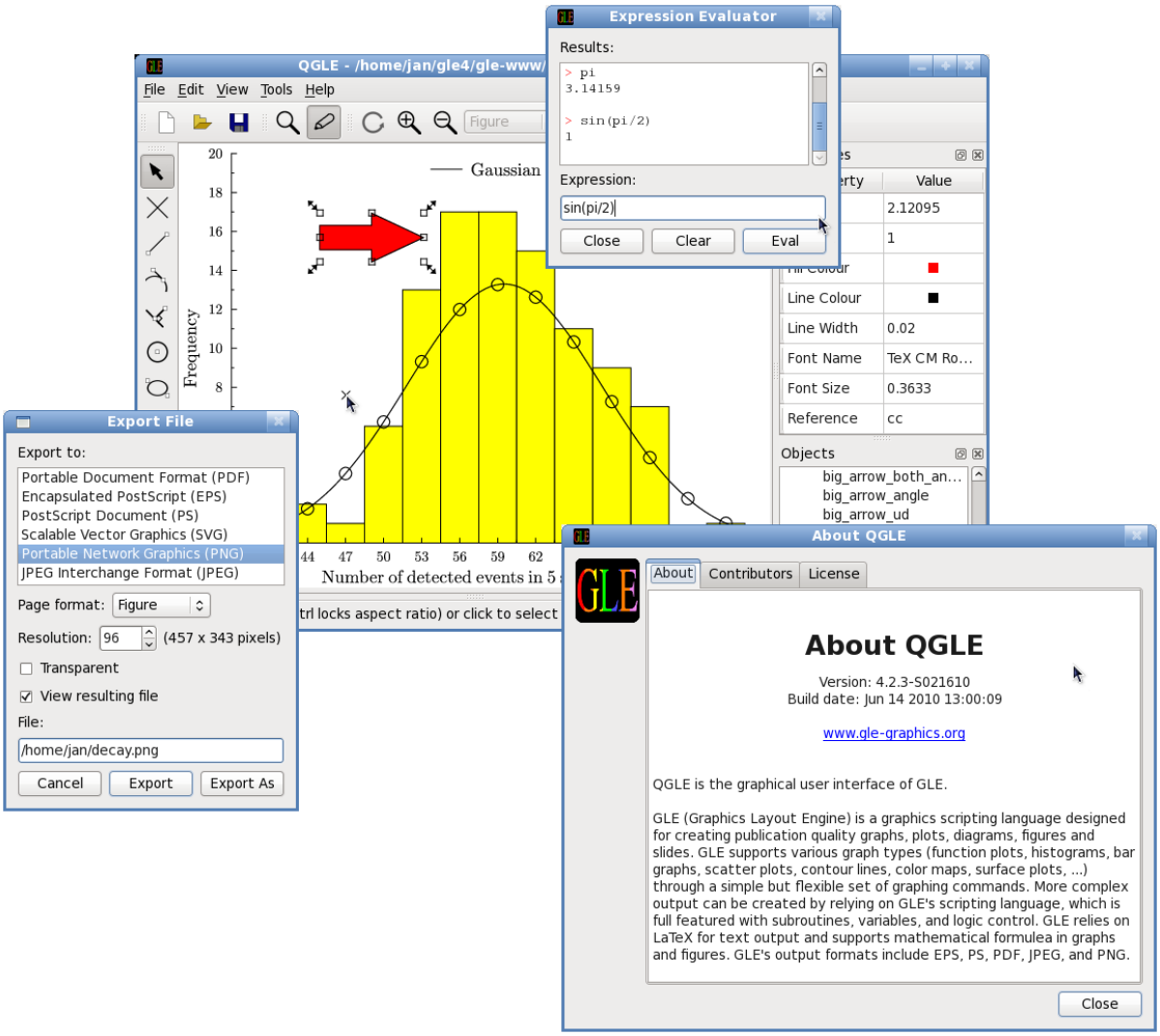


Figure 8.1: A screenshot of QGLE.

Chapter 9

Surface and Contour Plots

9.1 Surface Primitives

9.1.1 Overview

Surface plots three dimensional data using a wire frame with hidden line removal.
The simplist surface code would look like this.

```
begin surface
   data "myfile.z" 5 5
end surface
```

The surface block can contain the following commands:
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 θ ϕ 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*]

9.1.2 Surface Commands

size x y

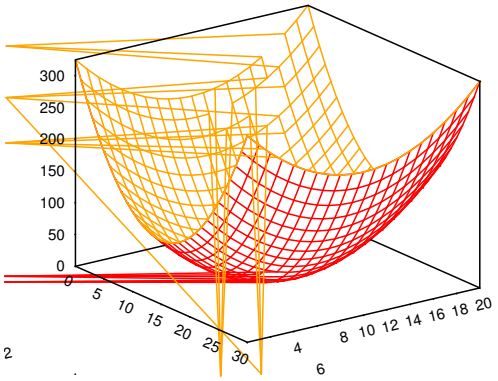
Specifies the size in cm to draw the surface. The 3d cube will fit inside this box. The default is 18cm x 18cm e.g. `size 10 10`

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 7 7
   data "jack.z"
   cube zlen 13
   top color orange
   underneath color red
end surface
```



data *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.

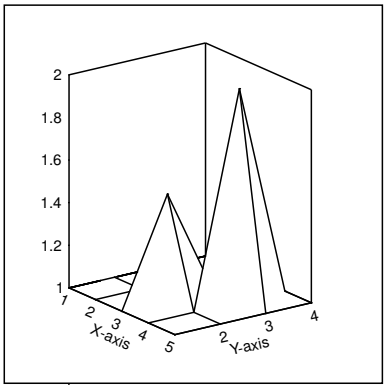
```
! nx 10  ny 20    xmin 1 xmax 10 ymin 1 ymax 20
1 2 42 4 5 2 31 4 3 2 4
1 2 42 4 5 2 31 4 3 2 4 etc...

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.
sample	Sets both xsample and ysample

(see also POINTS)

```
begin surface
   size 5 5
   xtitle "X-axis"
   ytitle "Y-axis"
   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.

ylines off
Stops SURF from drawing lines of constant Y.

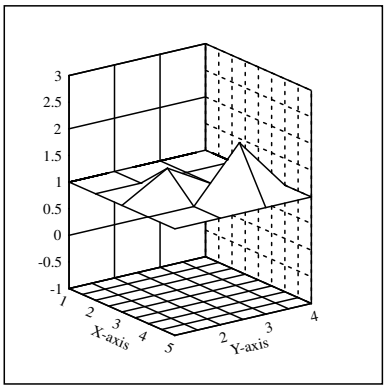
xaxis [min v] [max v] [step v] [color c] [lstyle l] [hei v] [off]

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 0.5 ystep 0.5
   back ystep 1 zstep 1
   right xstep 0.5 zstep 0.5 lstyle 2
   xtitle "X-axis" hei 0.3
   ytitle "Y-axis" hei 0.3
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 θ ϕ x

```
rotate 10 20 3
```

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.

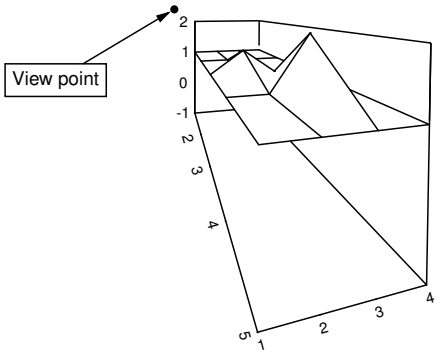
The third number is currently ignored.

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 infinitiy. 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 0.7
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 happenning.

base [xstep v] [ystep v] [lstyle l] [color c] [nohidden]
: Draws a grid on the base of the cube.

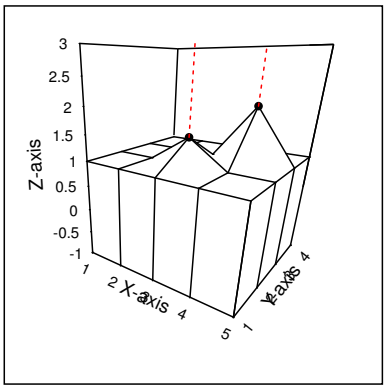
By default hidden lines are removed but NOHIDDEN will stop this from happenning.

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 happenning.

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 "X-axis"
   ytitle "Y-axis"
   ztitle "Z-axis"
   points "surf3.dat"
   riselines lstyle 2
   marker fcircle
   skirt on
   rotate 60 35 0
   view 2.5 3 0.6
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*.

9.2 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 jack.z now contains the required data. The resulting file can be used to generate surface plots with the begin/end surface block discussed in the previous section.

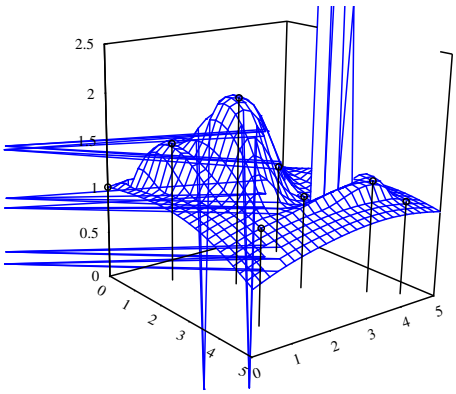
9.3 Fitz

FITZ fits smooth curves based on a set of 3D data points. E.g., given some data points (note that each line has three values: an x, y, and z coordinate):

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

Fitz creates a ".z" file that can be used in a surface block, a colormap or contour plot. The following example illustrates this.

```
begin fitz
   data "fitz.dat"
   x from 0 to 5 step 0.2
   y from 0 to 5 step 0.2
   ncontour 6
end fitz

begin surface
   size 7 7
   data "fitz.z"
   top color blue
   xaxis min 0 max 5 step 1
   yaxis min 0 max 5 step 1
   points "fitz.dat"
   droplines lstyle 1
   marker circle
   view 2.5 3 0.3
   harray 5000
end surface
```

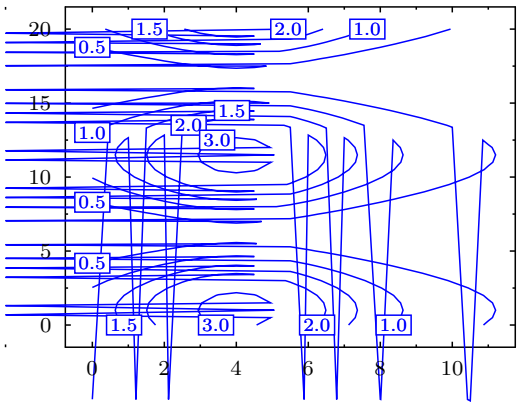


9.4 Contour

The contour block produces contour lines of a function $z = f(x, y)$.
The function $f(x, y)$ is given by a .z file. The .z file format is discussed on page 70. Recall that a .z file can be created from sample data points, that is (x, y, z) tuples, with the fitz block (Section 9.3), or from an implicit definition of $f(x, y)$ with a letz block (Section 9.2).

```
include "contour.gle"

begin contour
   data "saddle.z"
   values 0.5 1 1.5 2 3
end contour

begin graph
   title "Saddle Plot Contour Lines"
   data  "saddle-cdata.dat"
   d1 line color blue
end graph

contour_labels "saddle-clabels.dat" "fix 1"
```

Saddle Plot Contour Lines



The contour block can contain the following commands:

data *file$*
: Specifies the name of the .z file.

values $v_1, \ldots, v_n$
: Specifies the z-values to contour at.

values from v_1 to v_n step s
: Specifies the z-values to contour at by means of from/to/step.

smooth *integer*
: Specifies the smoothing parameter.

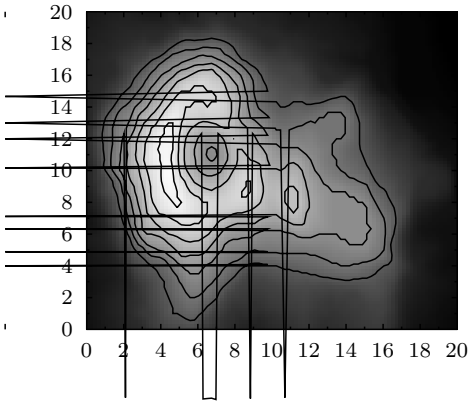
The contour block creates the files "data-clabels.dat" and "data-cdata.dat" with the prefix "data" the name of the .z file. The file "data-clabels.dat" contains information for drawing labels on the contour plot. This is done by the subroutine contour_labels defined in the library "contour.gle" in the example above. The file "data-cdata.dat" contains the (x, y) values of the contour lines. This file can be used as input to a graph block and plotted with the "d1 line" command as shown in the example above.

9.5 Color Maps

Color maps plot a function $z = f(x, y)$ by mapping z to a color range. The following example combines a color map with a contour plot.

```
begin contour
   data "volcano.z"
   values from 130 to 190 step 10
end contour

begin graph
   title "Auckland's Maunga Whau Volcano"
   data "volcano-cdata.dat"
   xaxis min 0 max 20
   yaxis min 0 max 20
   d1 line color black
   colormap "volcano.z" 100 100
end graph
```

Auckland's Maunga Whau Volcano



The options to the colormap command are as follows:

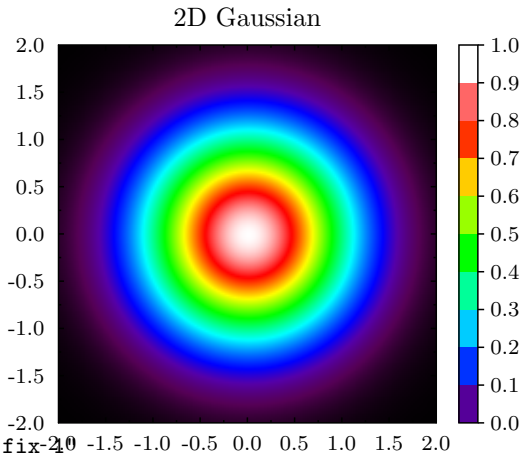
colormap *fct pixels-x pixels-y* [color] [invert] [zmin z_1] [zmax z_2] [palette *pal*]

- *fct* specifies the function to map. This can either be the name of a .z file, or it can be a function definition $f(x, y)$.

- *pixels-x*, *pixels-x* specify the dimension of the color map. A color map is a stored as a bitmap image and (*pixels-x*, *pixels-x*) are the resolution of this bitmap. A larger resolution yields more detail, but at the cost of longer computation time and a larger file size.
- *color* is an optional argument and indicates that the color map should be drawn in color (as opposed to grayscale).
- *invert* is an optional argument that inverts the color map. That is, large function values will be drawn in black and small function values in white.
- *zmin*, *zmax* are optional arguments that specify the range of the function.
- *palette* is an optional argument that specifies the palette to use. A palette is a subroutine that maps z values to colors. A number of example palette subroutines are included in the library "color.gle".

The following example is a color map of a two dimensional Gaussian.

```
include "color.gle"

sub gauss x y
   s = 0.75
   return exp(-(x^2+y^2)/(2*s^2))
end sub

begin graph
   title "2D Gaussian"
   xaxis min -2 max 2
   yaxis min -2 max 2
   colormap gauss(x,y) 200 200 zmin 0 zmax 1 color
end graph

amove xg(xgmax)+0.3 yg(ygmin)
color_range_vertical zmin 0 zmax 1 zstep 0.1 format "fix 1"
```



Chapter 10

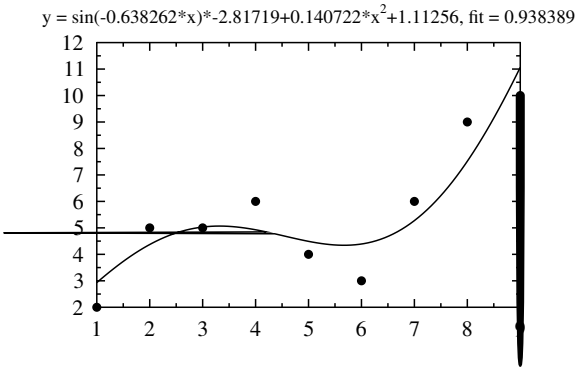
GLE Utilities

10.1 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 devided by n = 0.938389
y = sin(-0.638262*x)*-2.81719+0.140722*x^2+1.11256
```



10.2 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.

10.2.1 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 there parameters. A range is a rectangular section of your spreadsheet. A range can ether 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 toppics via the HELP command e.g. "HELP COPY"

10.2.2 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 [C*n*] or [R*n*]
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 nrows*
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 C*n*C*n* — R*n*R*n*

10.2.3 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 rangers are increased even if the number is not coppied 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 deleteing 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 usefull for entering data. After typing in `"% DATA c1c3"` or `"% DATA C2"` you can then enter data and pressing ¡cr¿ 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 seperated 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 usefull if your data file is very big and you don't want to waste diskspace 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 seperated 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 usefull if your data file is very big and you don't want to waste diskspace 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 commmand will load the 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 coppied 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 significat 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 coppied 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 coloumn, putting the partial sum's into another coloumn. e.g. `1,2,3,4` becomes `1,3,6,10`.

GENERATE [*pattern*] [*destination*]
: For generating a patter 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

△ triangle	● fcircle	⊙ odot	❀ flower	✍ handpen
△ wtriangle	◇ diamond	⊖ ominus	♣ club	✉ letter
▲ ftriangle	◇ wdiamond	⊕ oplus	 heart	☎ phone
□ square	◆ fdiamond	⊗ otimes	spade	✈ plane
□ wsquare	× cross	★ star	dag	❍ scircle
■ fsquare	+ plus	✣ star2	ddag	❑ ssquare
⌾ circle	— minus	✯ star3	§ snake	△ trianglez
⌾ wcircle	✳ asterisk	✪ star4	• dot	♢ diamondz

A.2 Functions and Variables

Function name	Returns ...
`abs(exp)`	absolute value of expression
`acos(exp)`	arccosine
`acosh(exp)`	inverse hyperbolic cosine
`acot(exp)`	1/atan(exp)
`acoth(exp)`	1/atanh(exp)
`acsc(exp)`	1/asin(exp)
`acsch(exp)`	1/asinh(exp)
`arg(i)`	i-th command line argument
`arg$(i)`	i-th command line argument
`asec(exp)`	1/acos(exp)
`asech(exp)`	1/acosh(exp)
`asin(exp)`	arcsine
`asinh(exp)`	inverse hyperbolic sine
`atan(exp)`	arctan
`atanh(exp)`	inverse hyperbolic tangent
`atn(exp)`	same as ATAN(exp)
`cos(exp)`	cosine
`cosh(exp)`	hyperbolic cosine
`cot(exp)`	1/tan(exp)
`coth(exp)`	1/tanh(exp)
`csc()`	1/sin(exp)

csch(exp)	1/sinh(exp)
dataxvalue(ds,i)	x-value of i-th point in ds (p. 52)
datayvalue(ds,i)	y-value of i-th point in ds (p. 52)
date$()	current date e.g. "Tue Apr 09 1991"
device$()	available devices e.g. "HARDCOPY, PS"
getenv(str)	returns environment variable "str"
eval(str)	evaluates given GLE expression
exp(exp)	exponent
\expr(exp)	substitute result of evaluating "exp"
file$()	returns GLE file name (without extension)
fix(exp)	exp rounded towards 0
format$(exp,format)	format exp as specified in format (p. 52)
height(name$)	the height of the object name$
int(exp)	integer part of exp
left$(str$,exp)	left exp characters of str$
len(str$)	the length of str$
log(exp)	log to base e of exp
log10(exp)	log to base 10 of exp
nargs()	number of command line arguments
ndata(ds)	number of points in data set ds (p. 52)
not(exp)	logical not of exp
num1$(exp)	as above but with no spaces
num$(exp)	string representation of exp
pageheight()	the height of the page (from size command)
pagewidth()	the width of the page (from size command)
path$()	returns the directory where the script is located
pointx(pt)	the x value of named point pt
pointy(pt)	the y value of named point pt
pos(str1$,str2$,exp)	position of str2$ in str1$ from exp
ptx(pt)	the x value of named point pt (abbreviation for pointx)
pty(pt)	the y value of named point pt (abbreviation for pointy)
rgb(red,green,blue)	create color given RGB values
rgba(red,green,blue,alpha)	create color given RGB and alpha values
rgb255(red,green,blue)	create color given RGB values
rgba255(red,green,blue,alpha)	create color given RGB and alpha values
right$(str$,exp)	rest of str$ starting at exp
rnd(exp)	random number between 0 and exp
sdiv(x,y)	return x/y or 0 if y = 0
sec(exp)	1/cos(exp)
sech(exp)	1/cosh(exp)
seg$(str$,exp1,exp2)	str$ from exp1 to exp2
sgn(exp)	returns 1 if exp is positive, -1 if exp is negative
sin(exp)	sine of exp
sinh(exp)	hyperbolic sine
sqr(exp)	exp squared
sqrt(exp)	square root of exp
tan(exp)	tangent of exp
tanh(exp)	hyperbolic tangent
tdepth(str$)	the depth of str$ assuming current the font, size
theight(str$)	the height of str$ assuming current font, size
time$()	current time e.g. "11:44:27"
todeg(exp)	convert from radians to degrees

`torad(exp)`	convert from degrees to radians
`twidth(str$)`	the width of `str$` assuming current font, size
`val(str$)`	value of the string `str$`
`width(name$)`	the width of the object `name$`
`xbar(x,i)`	the absolute x coordinate of the i-th bar at point x on the graph
`xend()`	the x end point of a text string when drawn
`xg(xexp)`	converts units of last graph to abs cm.
`xg3d(x,y,z)`	converts units of last 3D graph to abs cm.
`xpos()`	the current x point
`xy2angle(dx,dy)`	convert rectangular coordinates to polar angle (in degrees)
`yend()`	the y end point of a text string when drawn
`yg(yexp)`	converts units of last graph to abs cm.
`yg3d(x,y,z)`	converts units of last 3D graph to abs cm.
`ypos()`	the current y point

Variable name	Returns ...
`pi`	3.14...
`xgmin`	the minimum x-coordinate of the graph
`xgmax`	the maximum x-coordinate of the graph
`xg2min`	the minimum x2-coordinate of the graph
`xg2max`	the maximum x2-coordinate of the graph
`ygmin`	the minimum y-coordinate of the graph
`ygmax`	the maximum y-coordinate of the graph
`yg2min`	the minimum y2-coordinate of the graph
`yg2max`	the maximum y2-coordinate of the graph

A.3 LaTeX Macros

There are several LaTeX like commands which can be used within text, they are:

`\‘ \’ \v \u \= \^`	Implemented TeX accents
`\. \H \~ \’’`	Implemented TeX accents
`^{}`	Superscript
`_{}`	Subscript
`\\`	Forced newline
`\_`	Underscore character
`\,`	0.5em (em = width of the letter ‘m’)
`\:`	1em space
`\;`	2em space
`\!`	-0.5em space
`\tex{expression}`	Any LaTeX expression
`\char{22}`	Any character in current font
`\chardef{a}{hello}`	Define a character as a macro
`\def\v{hello}`	Defines a macro
`\movexy{2}{3}`	Moves the current text point
`\glass`	Makes move/space work on beginning of line
`\it`	Switches to italic font
`\kern{-0.1em}`	Change inter character distance
`\ldots`	...
`\lineskip{.1}`	Sets the default distance between lines of text
`\linegap{-1}`	Sets the minimum required gap between lines
`\lower{0.1em}{hello}`	Lower the given text
`\parskip{0.1em}`	Set distance between paragraphs
`\raise{0.1em}{hello}`	Raise the given text
`\rm`	Switches to roman font
`\bf`	Switches to bold font
`\rule{2}{4}`	Draws a filled in box, 2cm by 4cm
`\setfont{rmb}`	Sets the current text font
`\sethei{.3}`	Sets the font height (in cm)
`\setstretch{2}`	Scales the quantity of glue between words
`\tt`	Switches to typewriter (fixed space) font

A.4 LaTeX Symbols

\AA	\bot	\leftrightarrow	\setminus
\AE	\bullet	\leq	\sharp
\Delta	\cap	\lhook	\sigma
\Downarrow	\cdot	\ll	\sim
\Gamma	\chi	\lor	\simeq
\Im	\circ	\mapsto	\smallint
\L	\clubsuit	\mapstochar	\smile
\Lambda	\coprod	\mid	\spadesuit
\Leftarrow	\cup	\minus	\sqcap
\Leftrightarrow	\dag	\mp	\sqcup
\O	\dagger	\mu	\sqsubseteq
\OE	\dashv	\nabla	\sqsupseteq
\Omega	\ddag	\natural	\ss
\P	\ddagger	\nearrow	\star
\Phi	\degree	\neg	\subset
\Pi	\delta	\neq	\subseteq
\Psi	\diamond	\ni	\succ
\Re	\diamondsuit	\not	\succeq
\Rightarrow	\div	\nu	\sum
\S	\downarrow	\nwarrow	\supset
\Sigma	\ell	\o	\supseteq
\Theta	\emptyset	\odot	\swarrow
\Uparrow	\epsilon	\oe	\tau
\Updownarrow	\equiv	\ointop	\theta
\Upsilon	\eta	\omega	\times
\Xi	\exists	\ominus	\top
\aa	\flat	\oplus	\triangle
\ae	\forall	\oslash	\triangleleft
\aleph	\frown	\otimes	\triangleright
\alpha	\gamma	\owns	\uparrow
\amalg	\geq	\parallel	\updownarrow
\approx	\gg	\partial	\uplus
\ast	\heartsuit	\perp	\upsilon
\asymp	\i	\phi	\varepsilon
\backslash	\imath	\pi	\varphi
\beta	\in	\pm	\varpi
\bigcap	\infty	\prec	\varrho
\bigcirc	\intop	\preceq	\varsigma
\bigcup	\iota	\prime	\vartheta
\bigodot	\j	\prod	\vdash
\bigoplus	\jmath	\propto	\vee
\bigotimes	\kappa	\psi	\wedge
\bigsqcup	\l	\rho	\wp
\bigtriangledown	\lambda	\rhook	\wr
\bigtriangleup	\land	\rightarrow	\xi
\biguplus	\leftarrow	\rightharpoondown	\zeta
\bigvee	\leftharpoondown	\rightharpoonup	
\bigwedge	\leftharpoonup	\searrow	

A.5 Installing GLE

A.6 Fonts

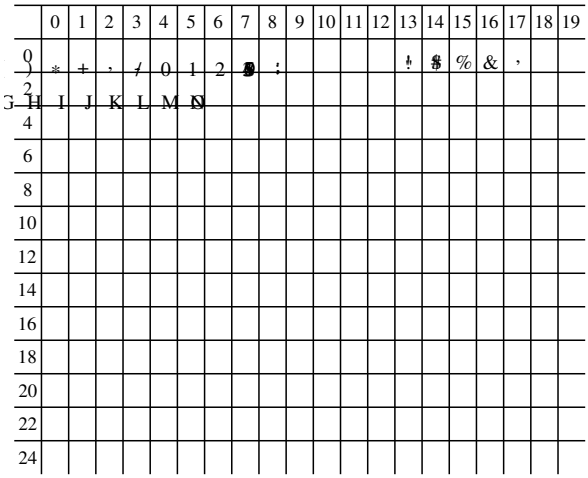
rm	Roman
rmb	**Roman Bold**
rmi	*Roman Italic*
ss	Sans Serif
ssb	**Sans Serif Bold**
ssi	*Sans Serif Italic*
tt	Typewriter
ttb	**Typewriter Bold**
tti	*Typewriter Italic*

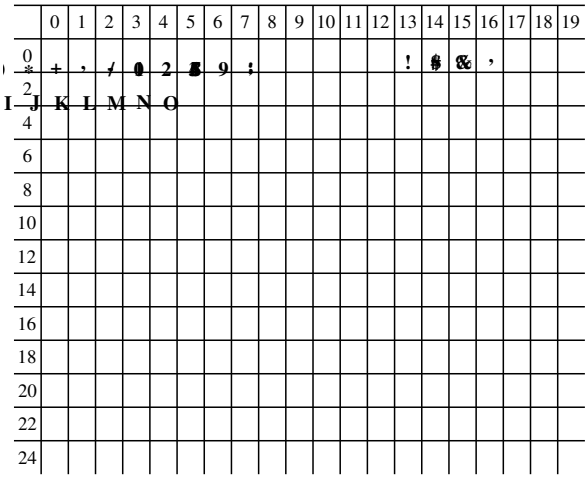
texcmr	Computer Modern Roman
texcmb	**Computer Modern Bold**
texcmti	*Computer Modern Text Italic*
texcmmi	*ComputerModernMathsItalic*
texcmss	Computer Modern Sans Serif
texcmssb	**Computer Modern Sans Serif Bold**
texcmssi	*Computer Modern Sans Serif Italic*
texcmtt	Computer Modern Typewriter Text
texcmitt	*Computer Modern Italic Typewriter*

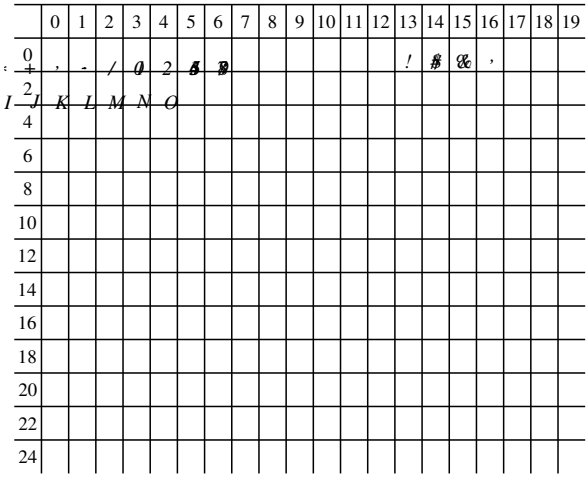
plsr	Simplex Roman
pldr	Duplex Roman
pltr	Triplex Roman
plti	*Triplex Italic*
plcr	Complex Roman
plci	*Complex Italic*
plss	*Simplex Script*
plcs	*Complex Script*
plsa	Simplex Ascii
plba	Block Ascii
plge	Gothic English
plgg	Gothic German
plgi	Gothic Italian
plsg	Τυπλεξ Ησελ

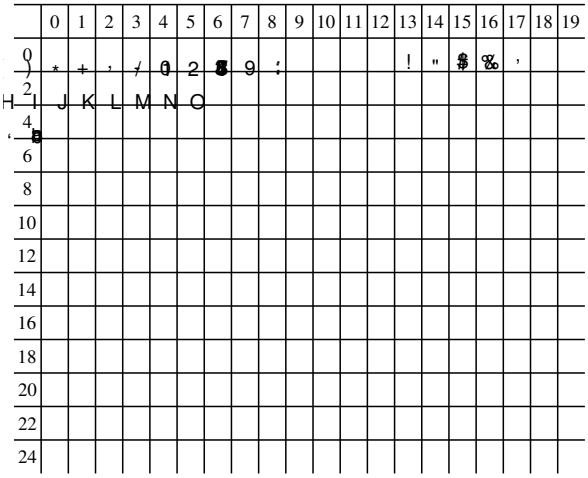
pstr	Times-Roman
pstb	**Times-Bold**
psti	*Times-Italic*
pstbi	***Times-BoldItalic***
psc	Courier
pscb	**Courier-Bold**
psco	*Courier-Oblique*
pscbo	***Courier-BoldOblique***
psh	Helvetica
pshb	**Helvetica-Bold**
psho	*Helvetica-Oblique*
pshbo	***Helvetica-BoldOblique***
pshc	Helvetica-Condensed
pshcb	**Helvetica-Condensed-Bold**
pshcdo	*Helvetica-Condensed-Oblique*
pshn	Helvetica-Narrow
pshnb	**Helvetica-Narrow-Bold**
pshno	*Helvetica-NarrowOblique*
pshnbo	***Helvetica-Narrow-BoldOblique***
psagb	AvantGarde-Book
psagd	**AvantGarde-Demi**
psagbo	*AvantGarde-BookOblique*
psagdo	***AvantGarde-DemiOblique***
psbl	Bookman-Light
psbd	**Bookman-Demi**
psbli	*Bookman-LightItalic*
psbdi	***Bookman-DemiItalic***
psncsr	NewCenturySchlbk-Roman
psncsb	**NewCenturySchlbk-Bold**
psncsi	*NewCenturySchlbk-Italic*
psncsbi	***NewCenturySchlbk-BoldItalic***
pspr	Palatino-Roman
pspb	**Palatino-Bold**
pspi	*Palatino-Italic*
pspbi	***Palatino-BoldItalic***
pssym	ΠοστΣχριπτ Συμβολ
pszcmi	*ZapfChancery-MediumItalic*
pszd	✺❁❒❊✤❉■✳✪❁▼▲

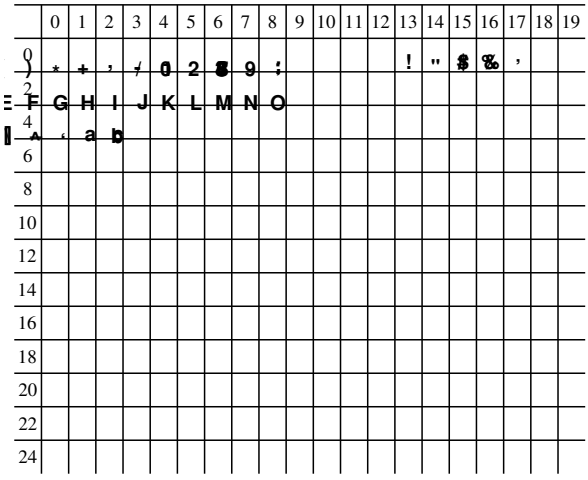
A.7 Font Tables

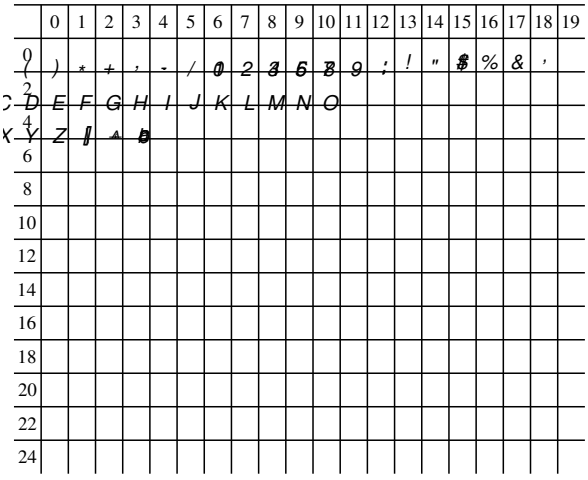


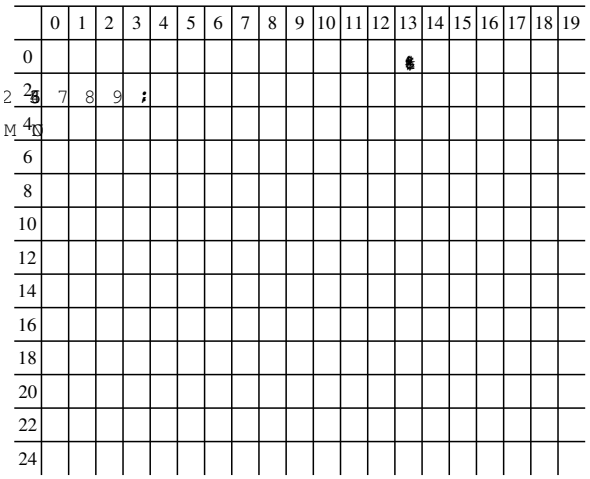


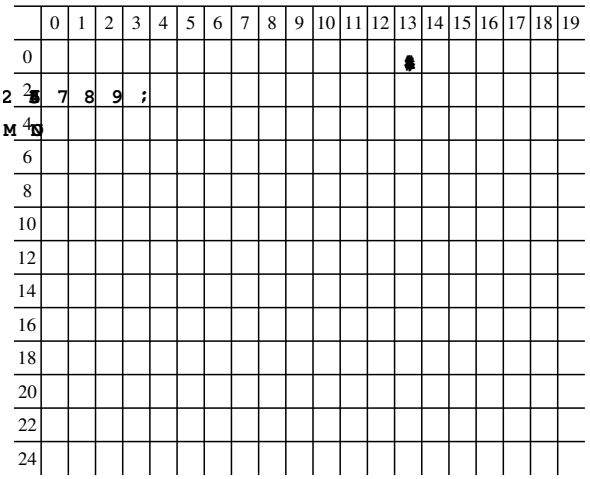


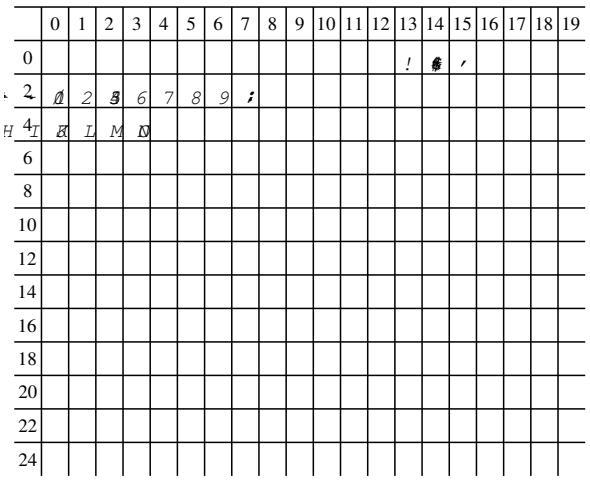




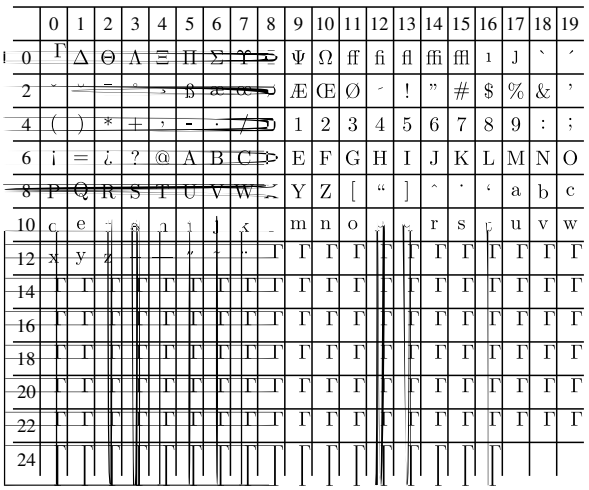




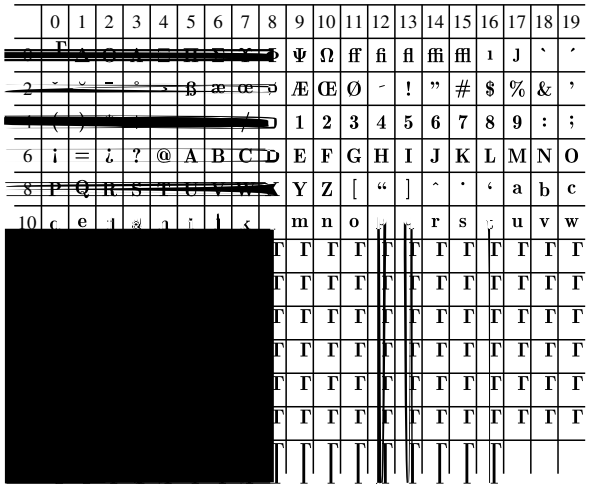




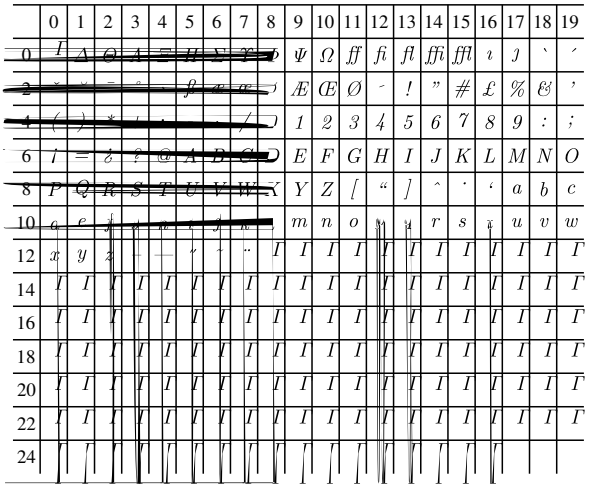
TEXComputer Modern Roman (texcmr)



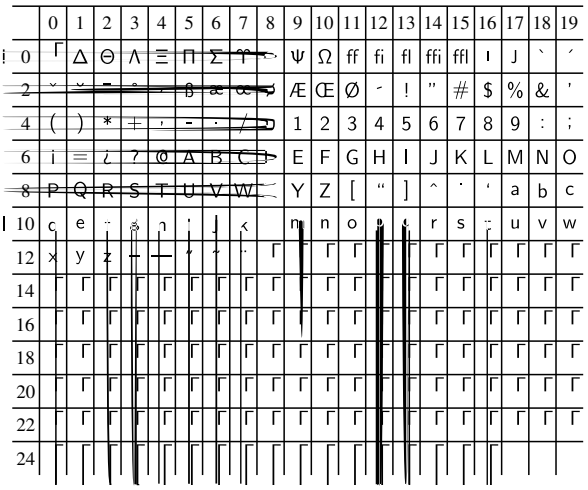
TEXComputer Modern Bold (texcmb)



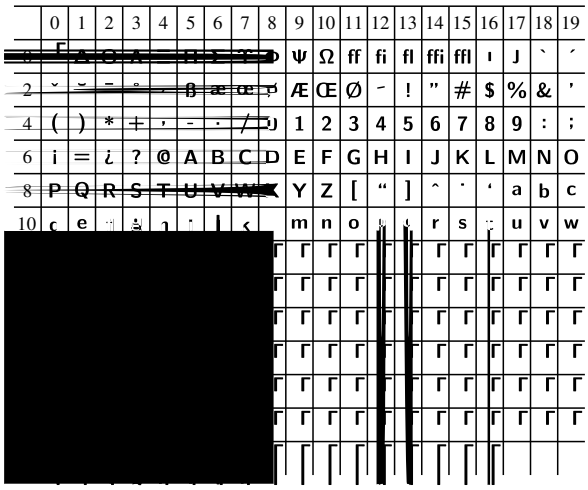
TEXComputer Modern Text Italic (texcmti)



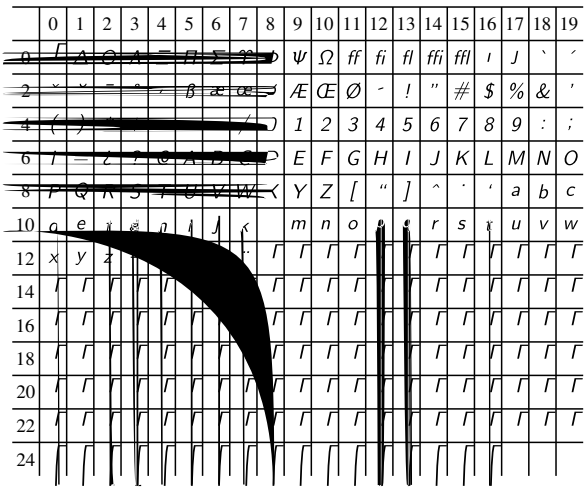
TEXComputer Modern Sans Serif (texcmss)



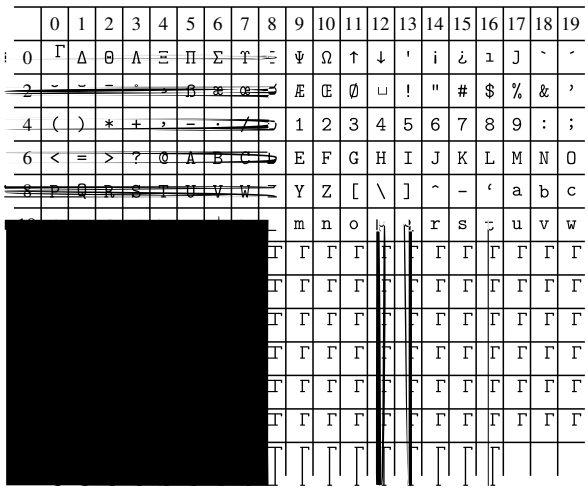
TEXComputer Modern Sans Serif Bold (texcmssb)



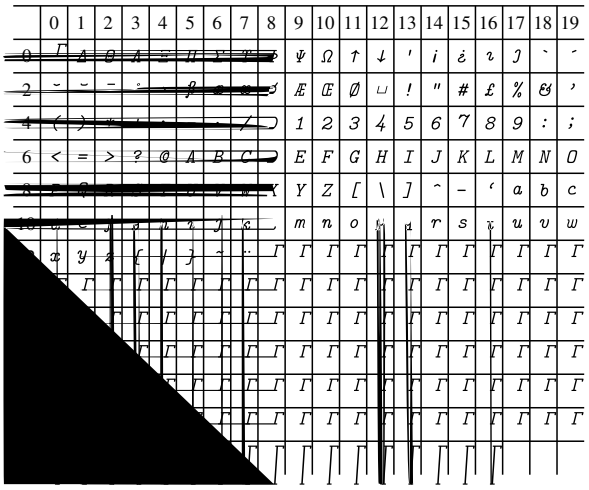
TEXComputer Modern Sans Serif Italic (texcmssi)



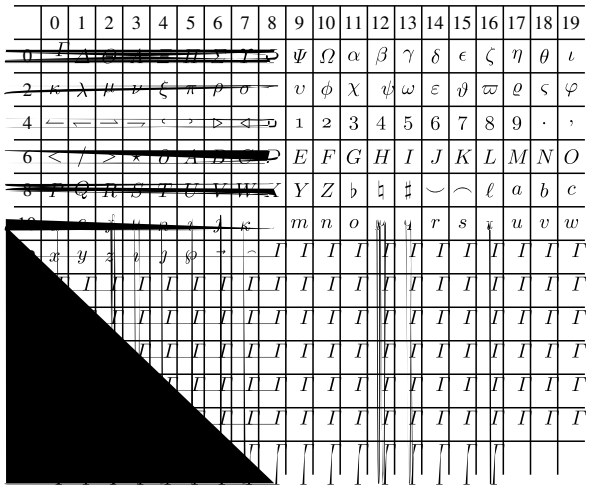
TEXComputer Modern Typewriter Text (texcmtt)



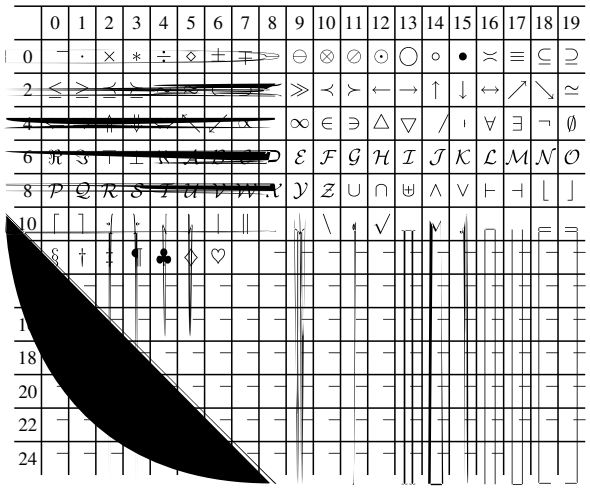
TEXComputer Modern Italic Typewriter (texcmitt)



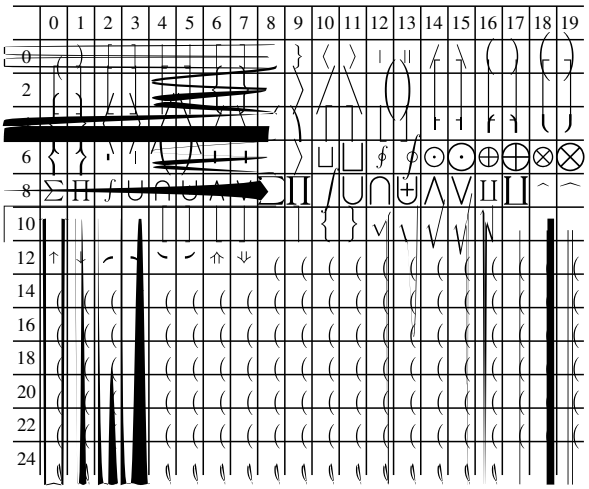
TEXComputer Modern Maths Italic (texcmmi)

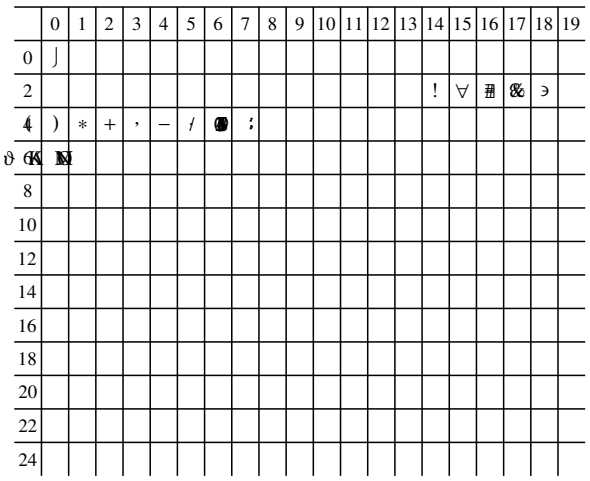


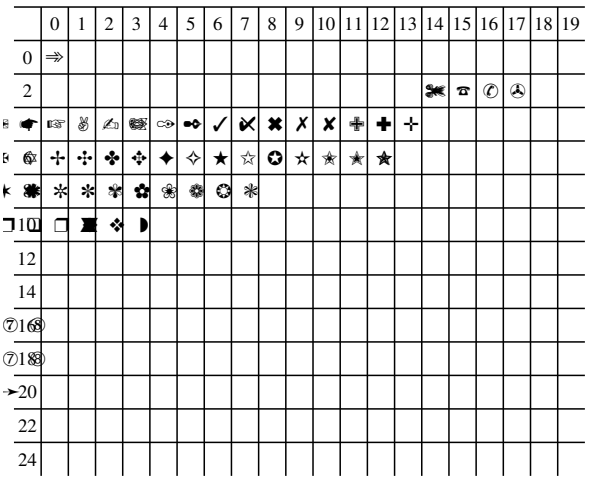
TEXComputer Modern Symbol (texcmsy)



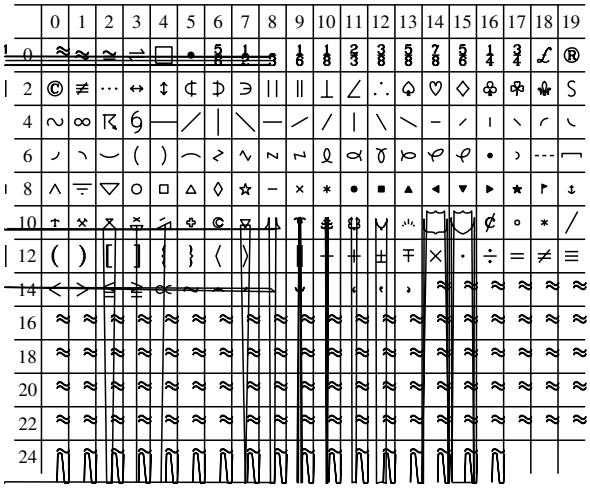
TEXComputer Modern Extensible (texcmex)



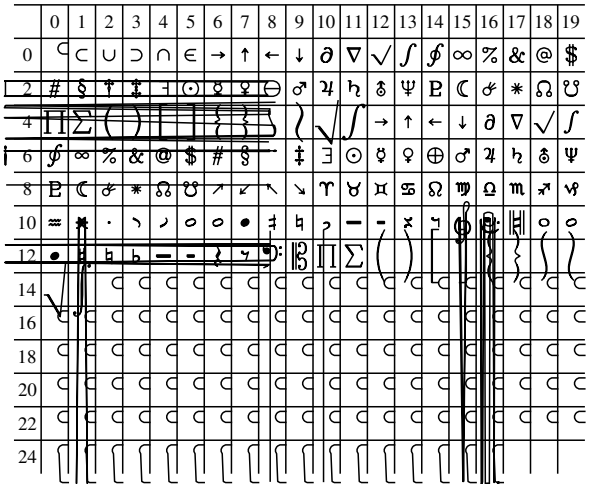


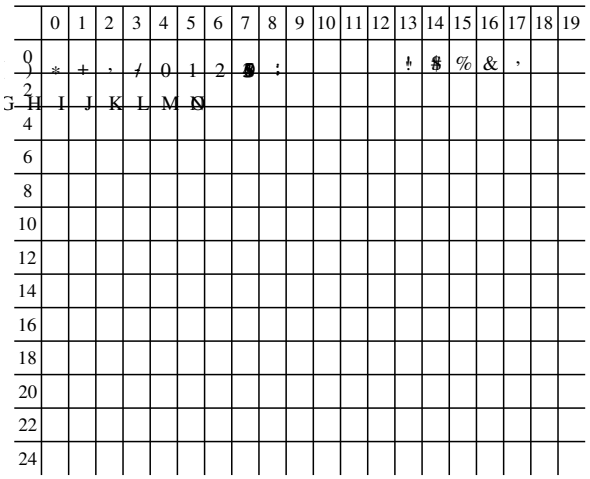


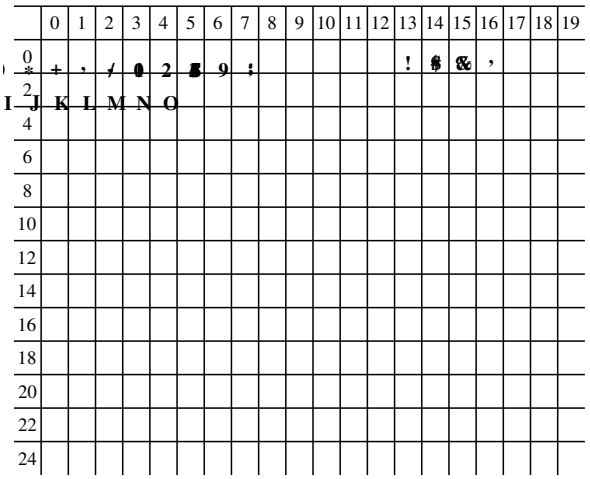
Plotter Symbols one (plsym1)

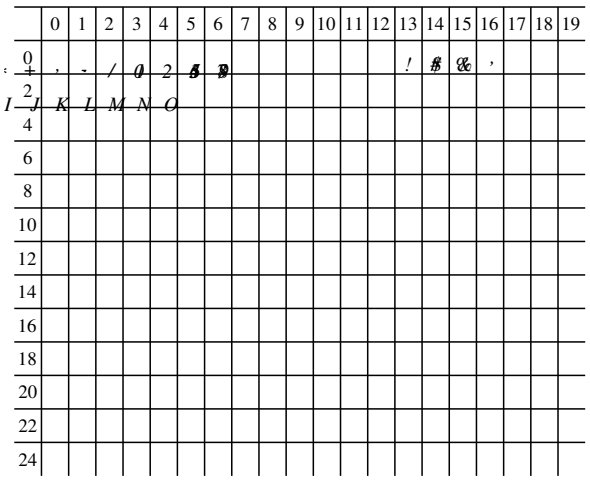


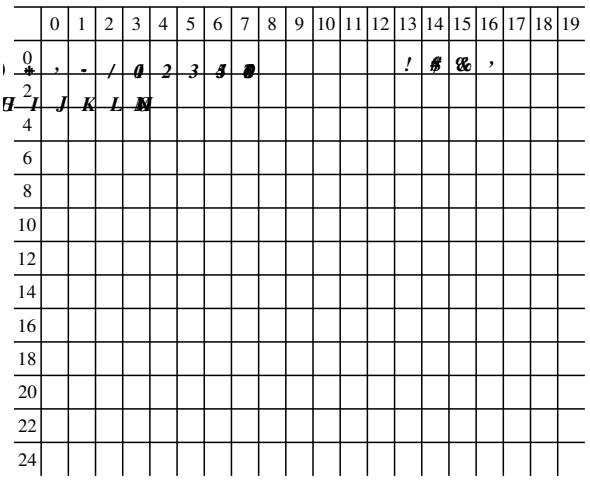
Plotter Symbols two (plsym2)

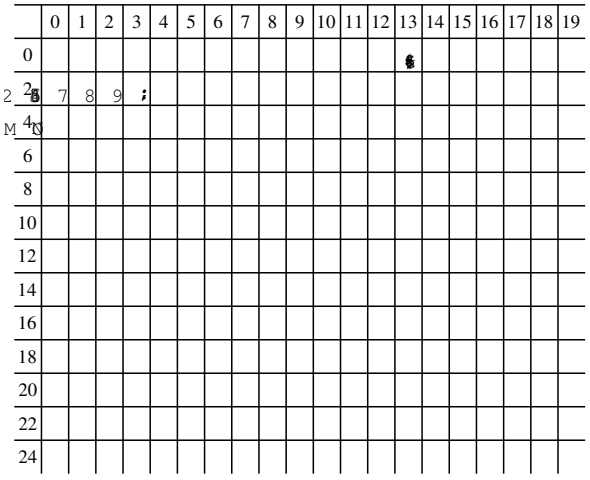


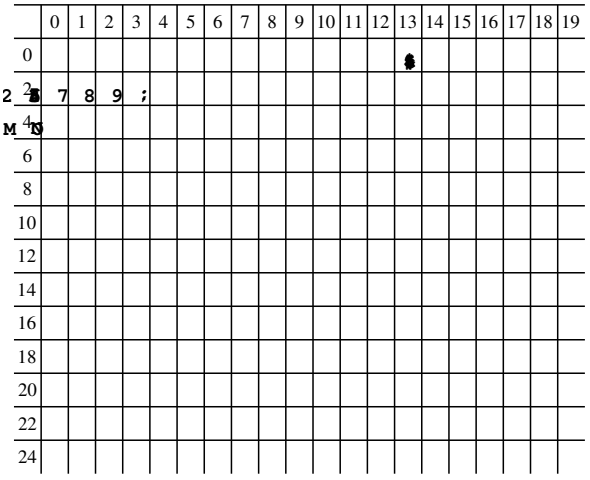


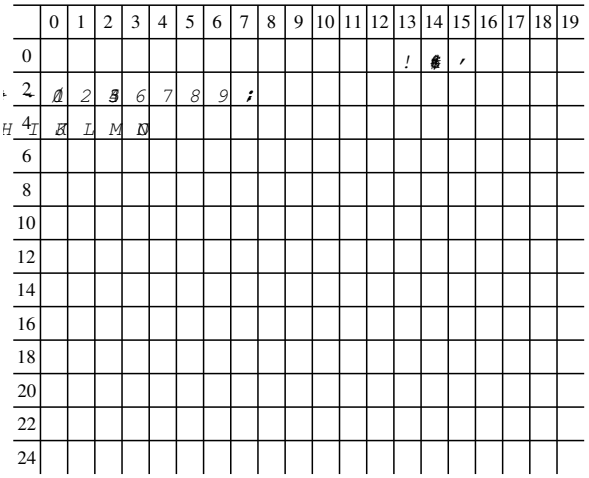


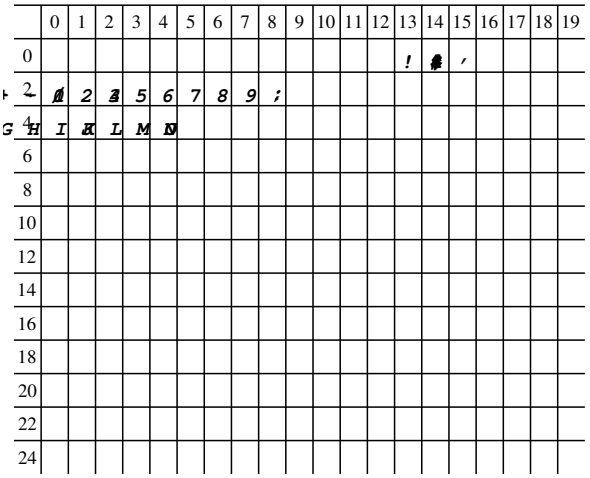


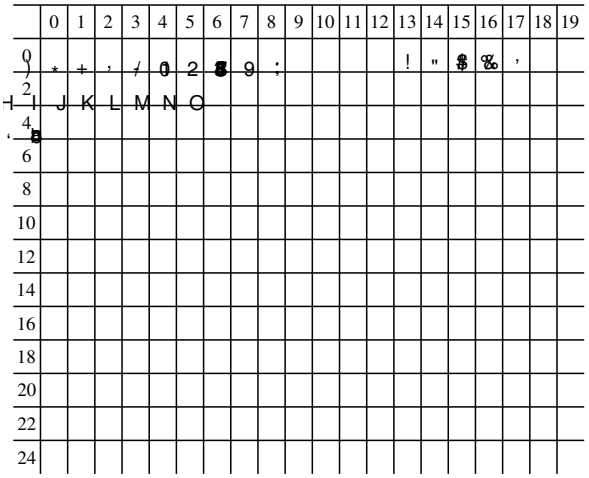


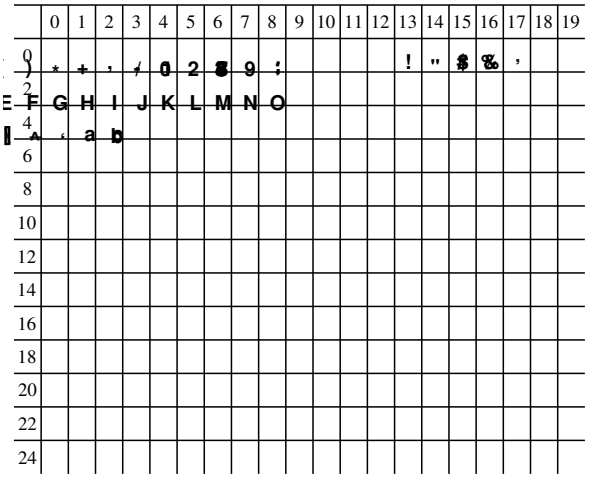


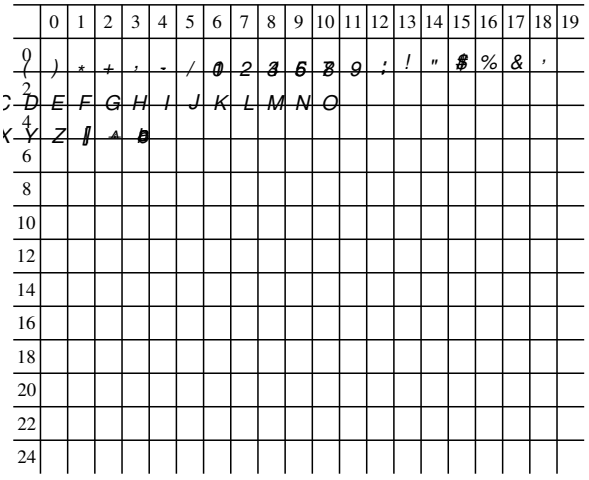


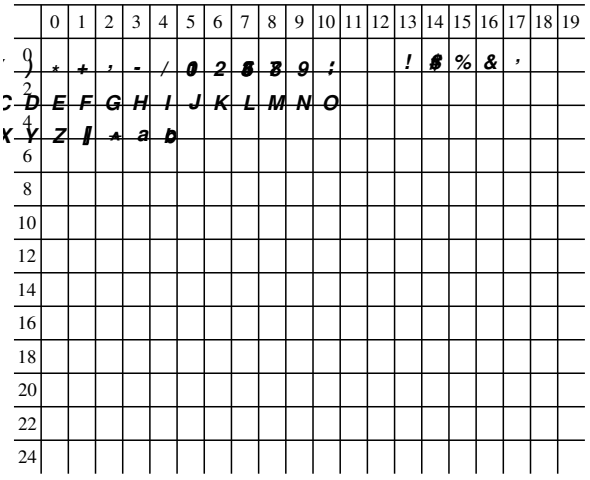


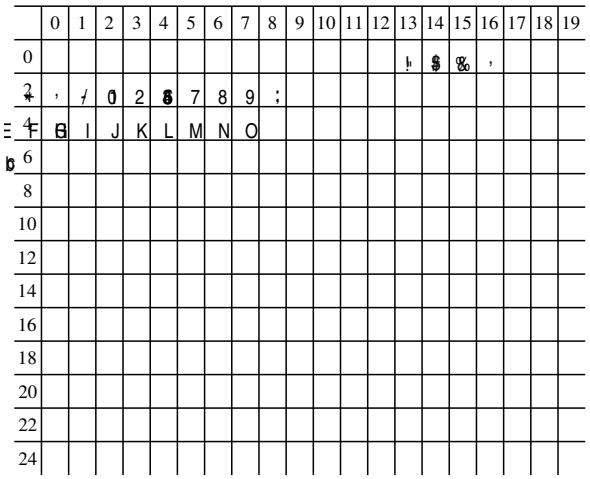


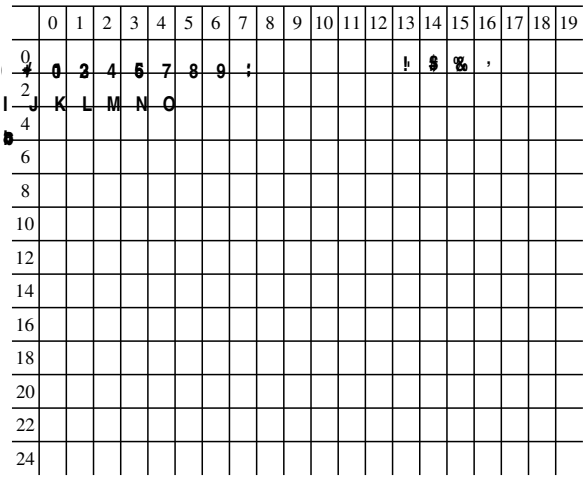


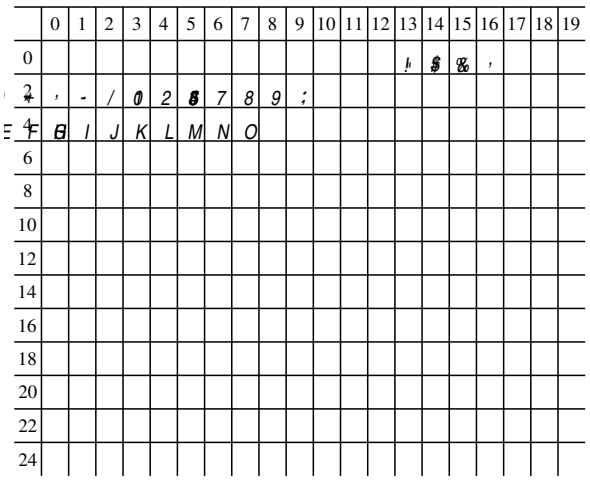


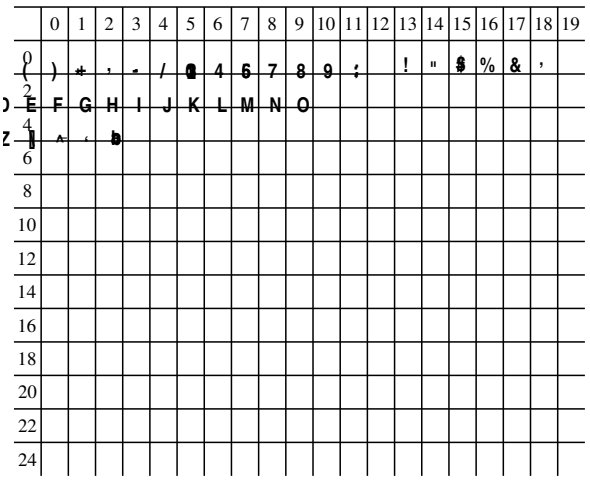


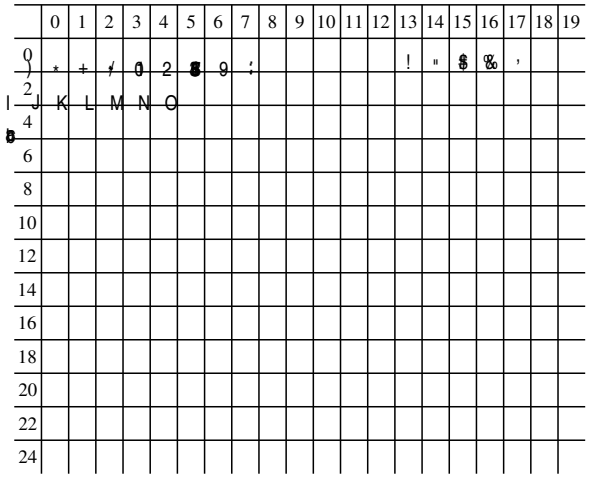


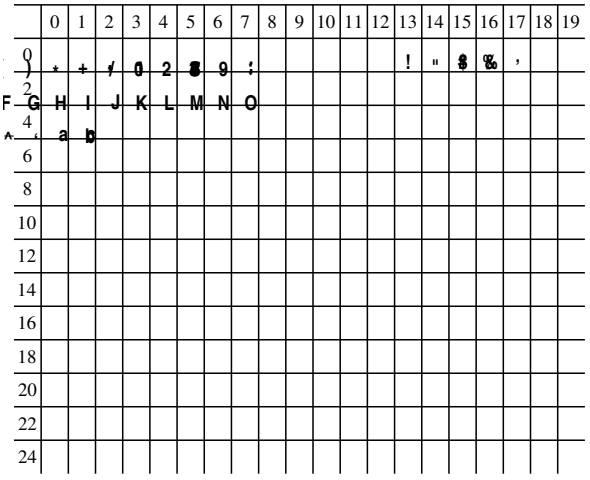


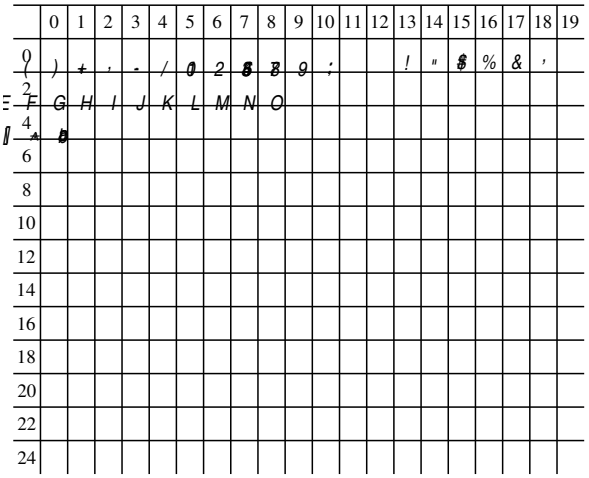


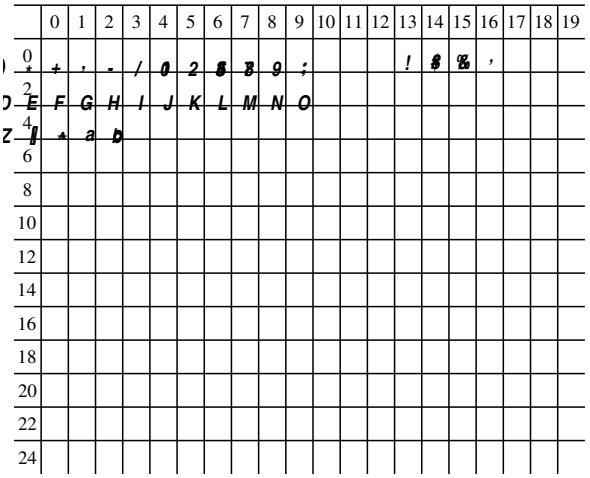


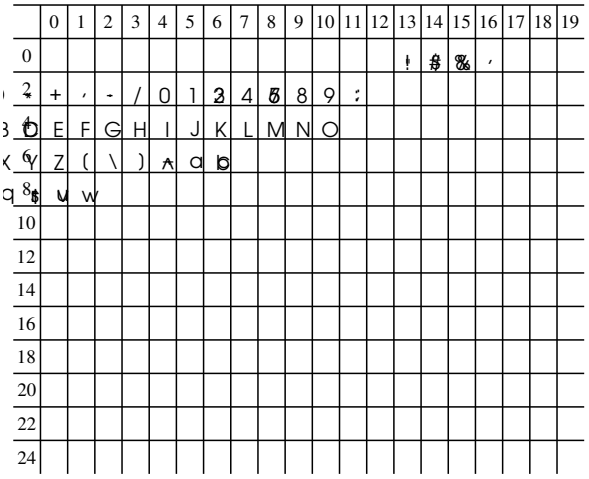


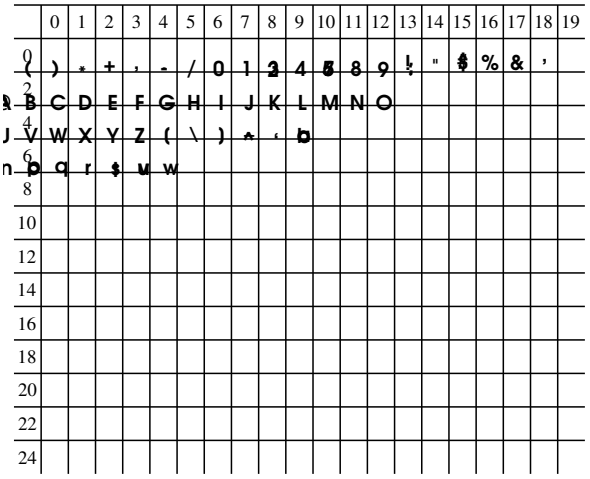


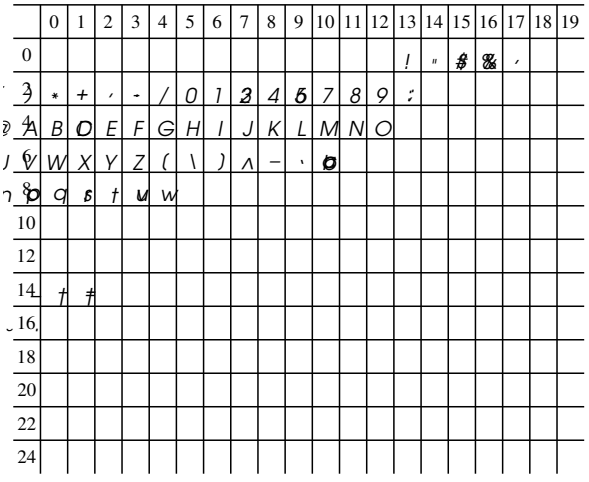


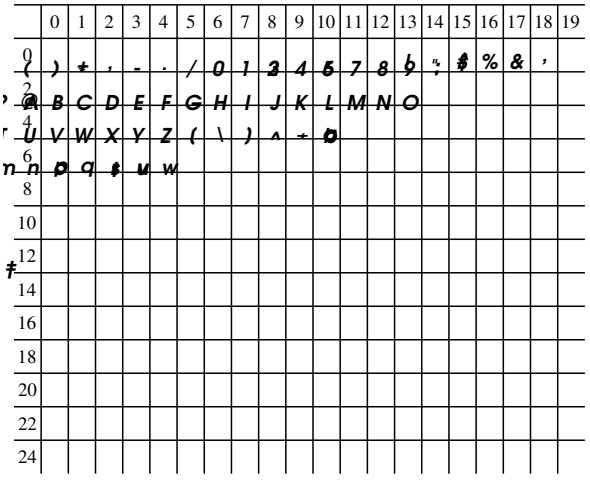


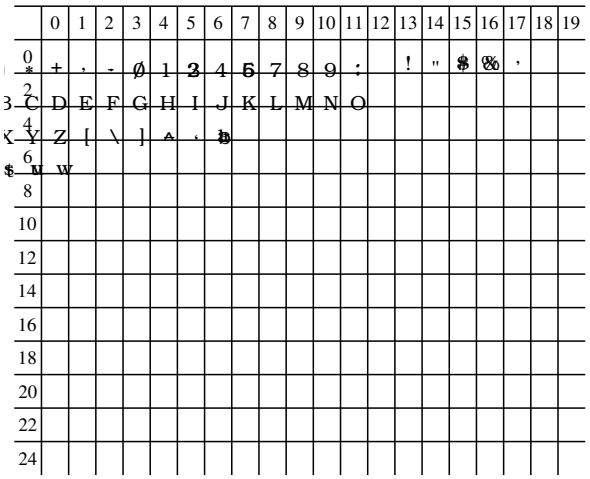


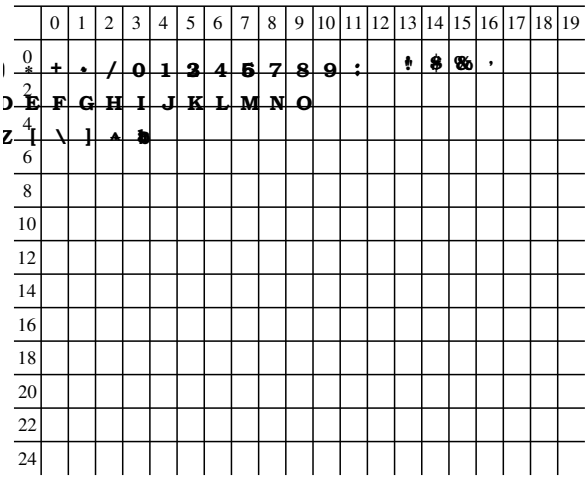


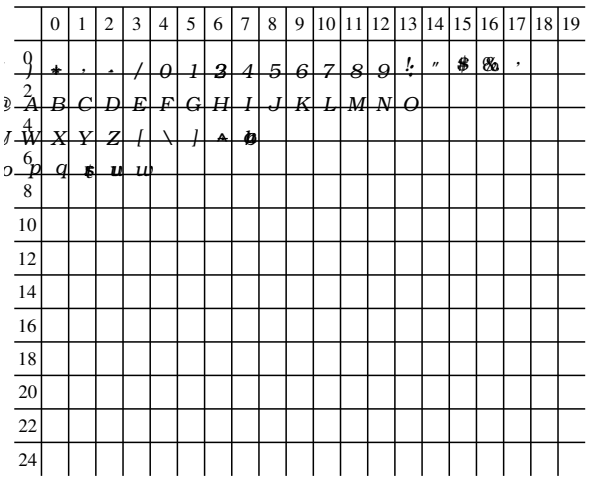


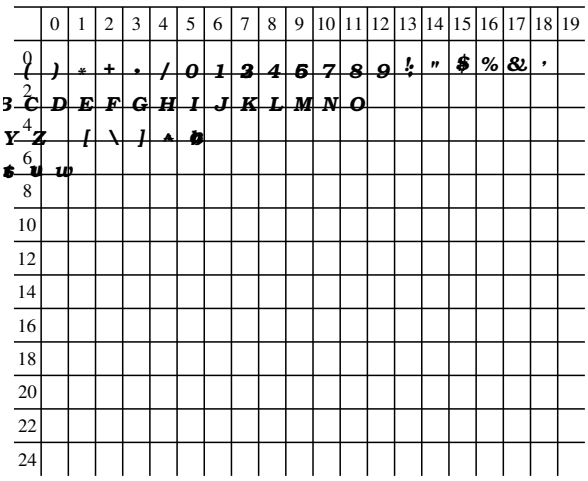


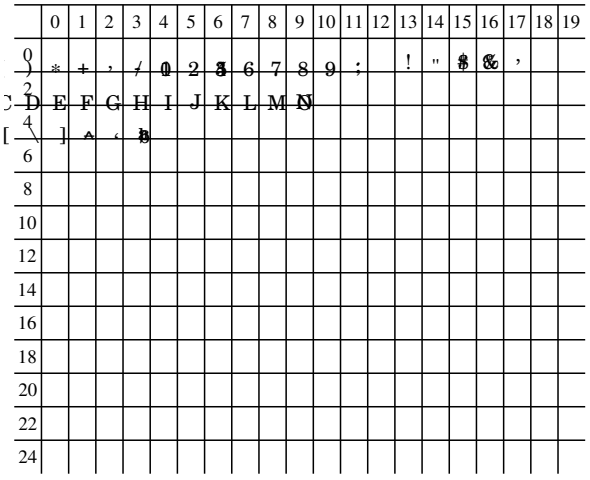


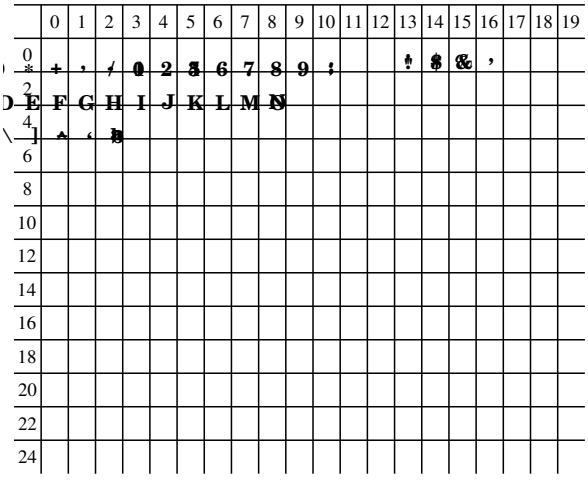


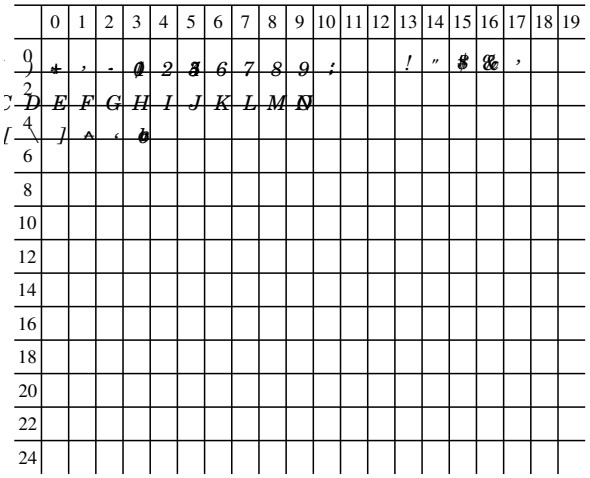


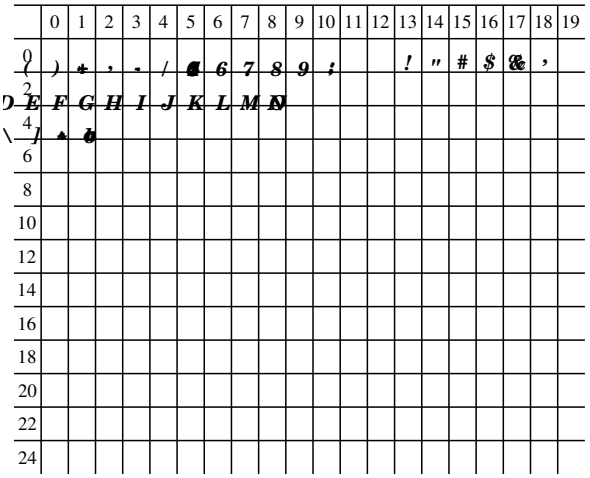


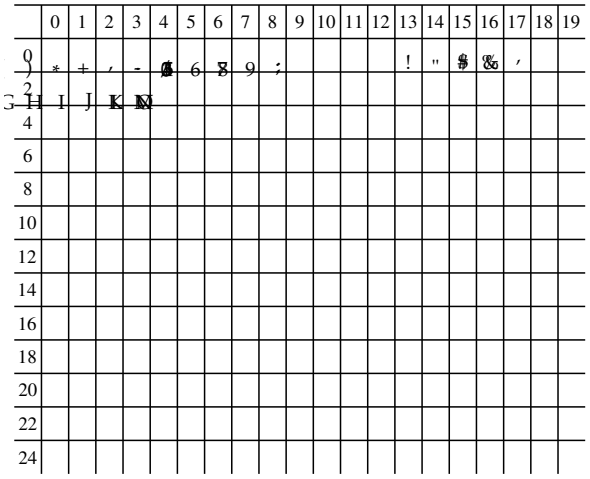


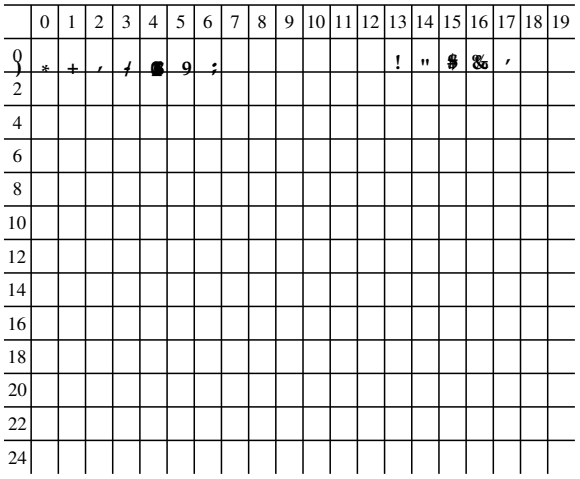


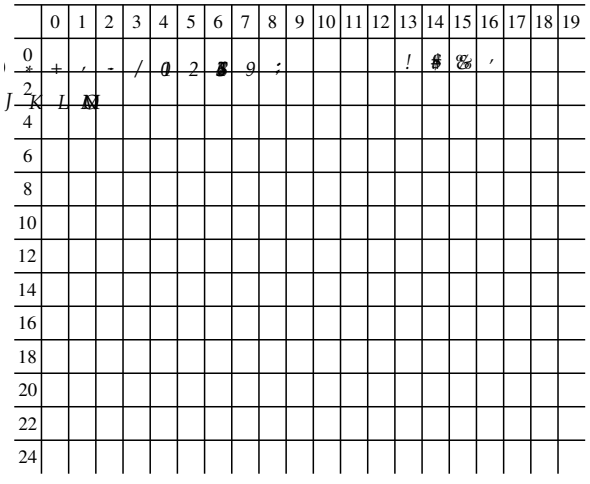


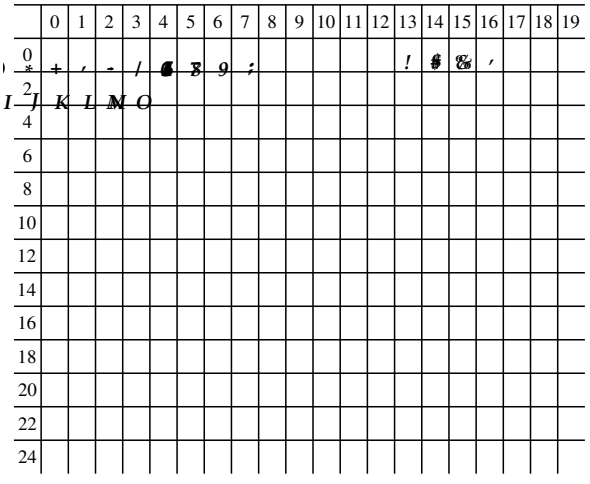


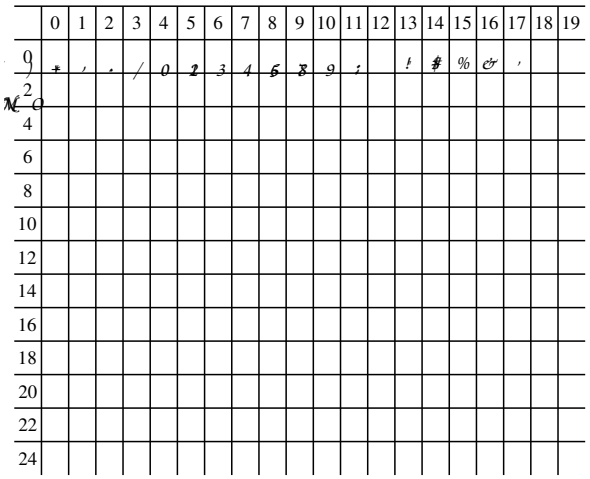




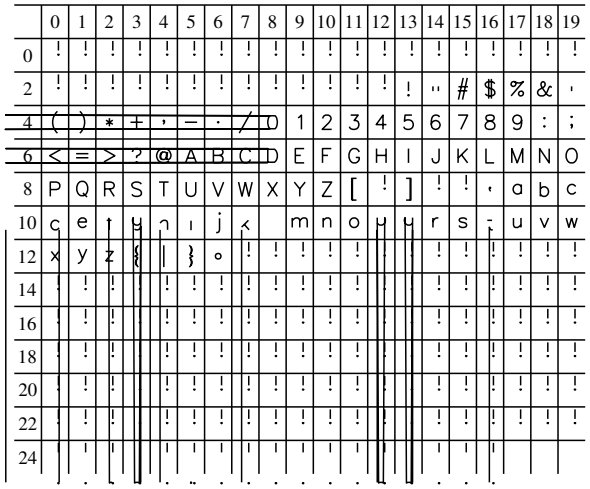




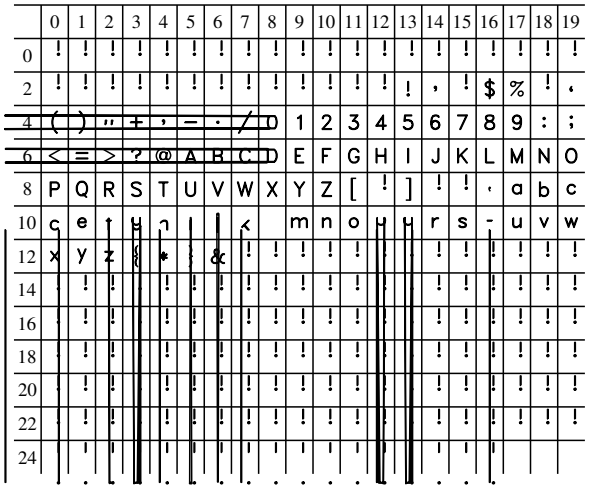




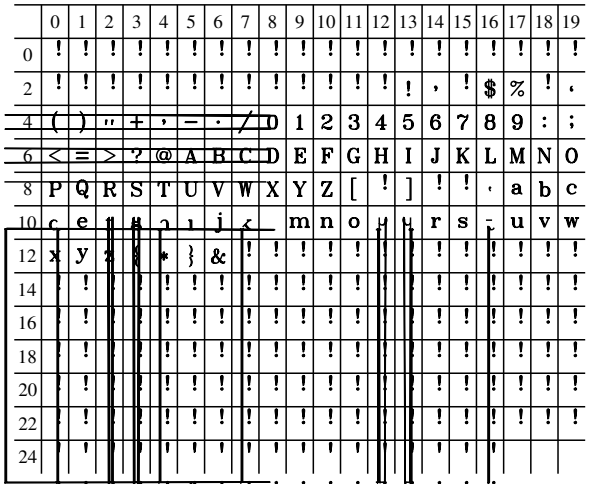
Plotter Simplex Roman (plsr)



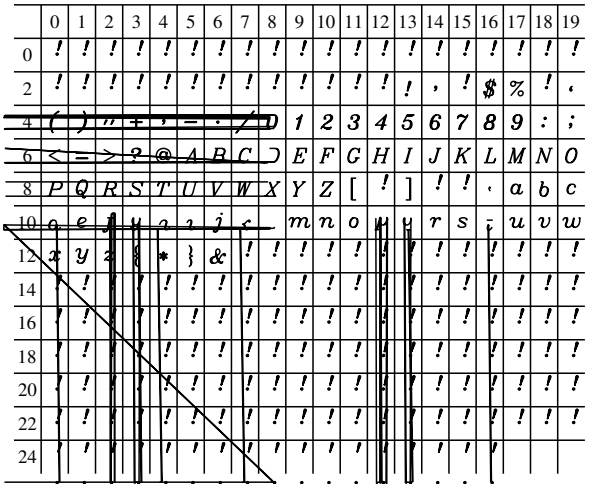
Plotter Duplex Roman (pldr)



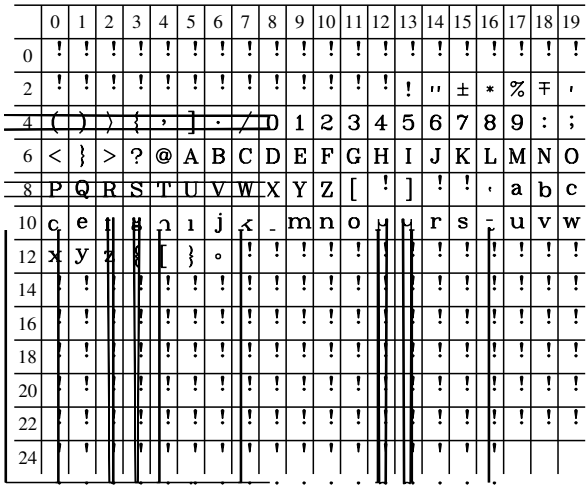
Plotter Triplex Roman (pltr)



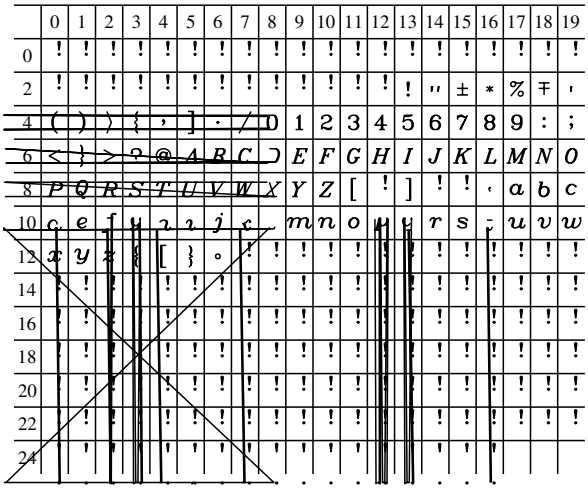
Plotter Triplex Italic (plti)



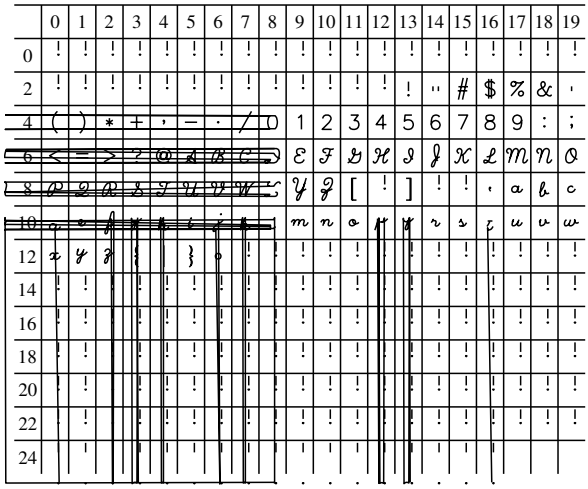
Plotter Complex Roman (plcr)



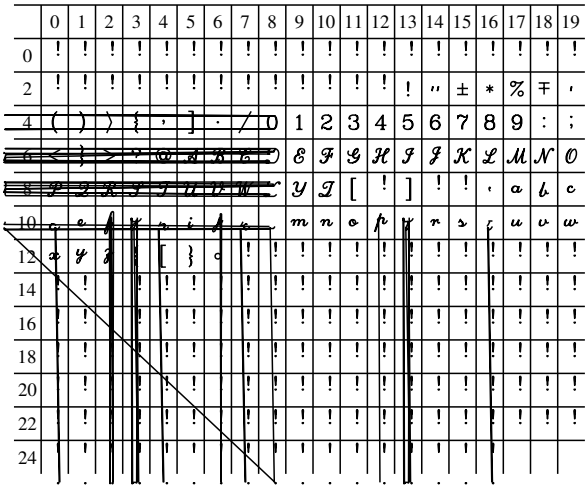
Plotter Complex Italic (plci)



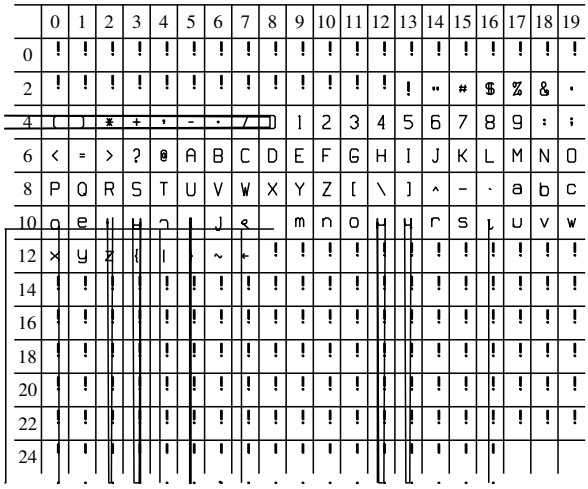
Plotter Simplex Script (plss)



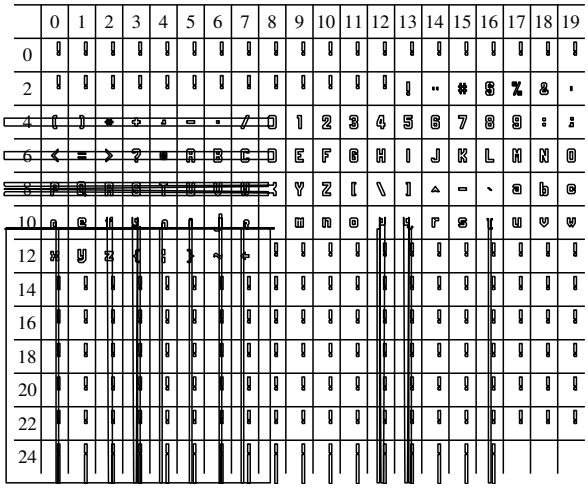
Plotter Complex Script (plcs)



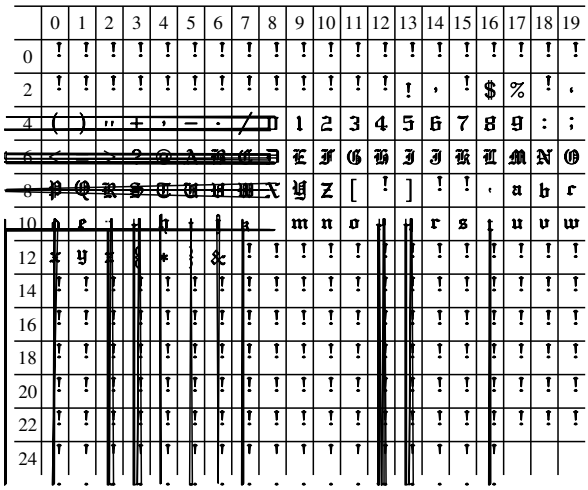
Plotter Simplex Ascii (plsa)



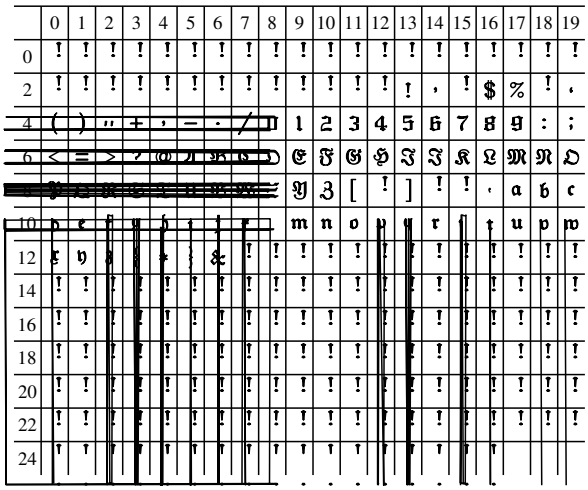
Plotter Block Ascii (plba)



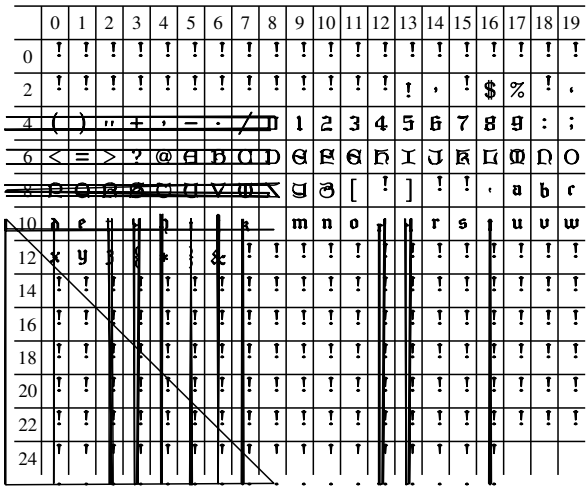
Plotter Gothic English (plge)



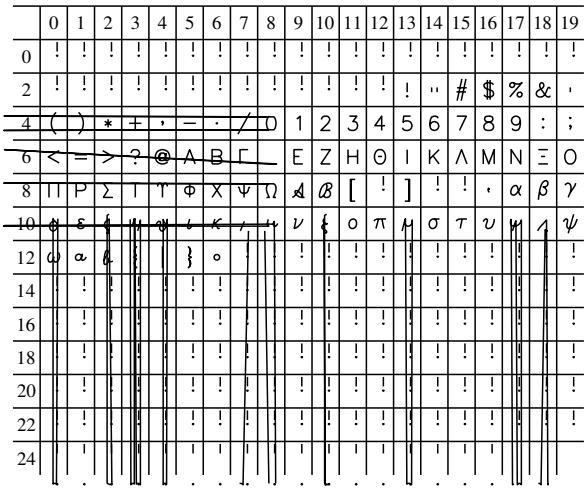
Plotter Gothic German (plgg)



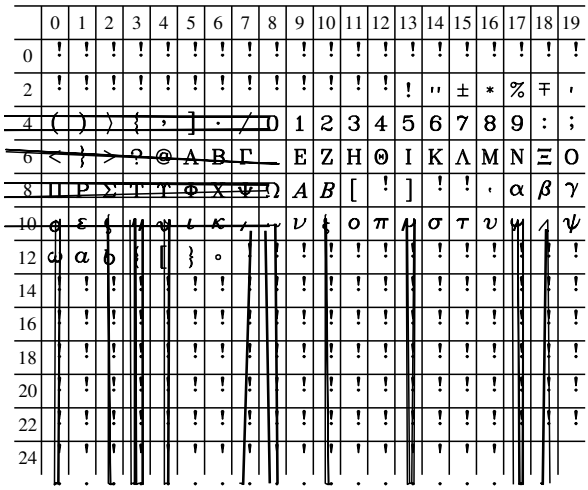
Plotter Gothic Italian (plgi)



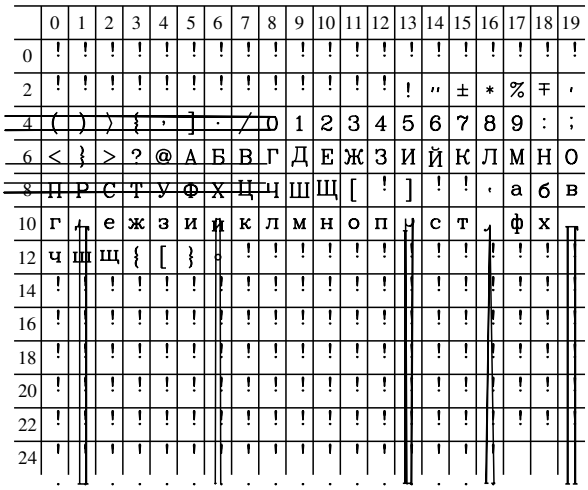
Plotter Simplex Greek (plsg)



Plotter Complex Greek (plcg)



Plotter Complex Cartographic (plcc)



A.8 Predefined Colors

GLE supports these SVG/X11 standard colors (sorted by color)

indianred
lightcoral
salmon
darksalmon
lightsalmon
crimson
red
firebrick
darkred
pink
lightpink
hotpink
deeppink
mediumvioletred
palevioletred
coral
tomato
orangered
darkorange
orange
gold
yellow
lightyellow
lemonchiffon
lightgoldenrodyellow
papayawhip
moccasin
peachpuff
palegoldenrod
khaki
darkkhaki
lavender
thistle
plum
violet
orchid
fuchsia
magenta
mediumorchid

mediumpurple
blueviolet
darkviolet
darkorchid
darkmagenta
purple
indigo
slateblue
darkslateblue
greenyellow
chartreuse
lawngreen
lime
limegreen
palegreen
lightgreen
mediumspringgreen
springgreen
mediumseagreen
seagreen
forestgreen
green
darkgreen
yellowgreen
olivedrab
olive
darkolivegreen
mediumaquamarine
darkseagreen
lightseagreen
darkcyan
teal
aqua
cyan
lightcyan
paleturquoise
aquamarine
turquoise
mediumturquoise

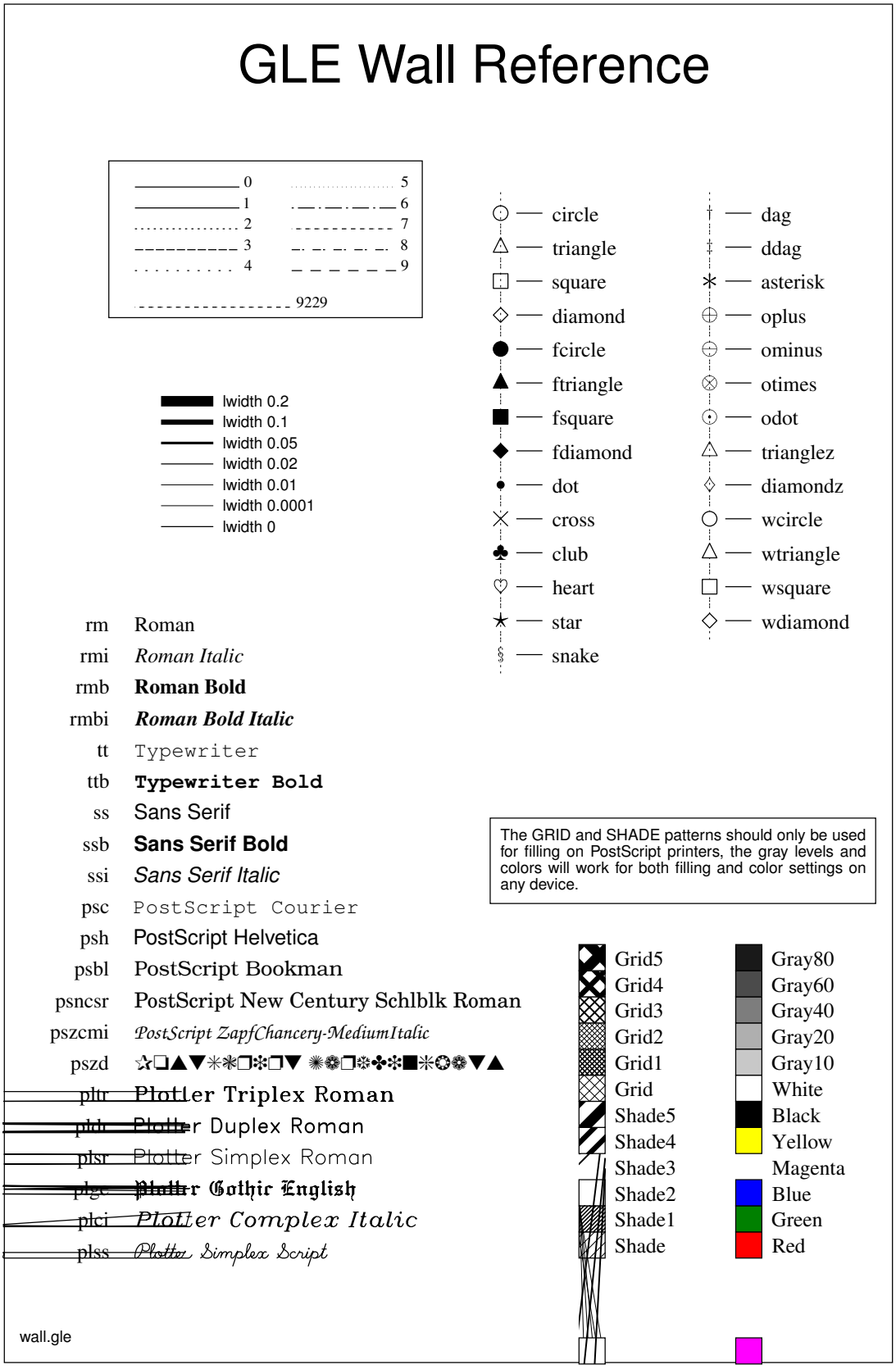
darkturquoise
cadetblue
steelblue
lightsteelblue
powderblue
lightblue
skyblue
lightskyblue
deepskyblue
dodgerblue
cornflowerblue
mediumslateblue
royalblue
blue
mediumblue
darkblue
navy
midnightblue
cornsilk
blanchedalmond
bisque
navajowhite
wheat
burlywood
tan
rosybrown
sandybrown
goldenrod
darkgoldenrod
peru
chocolate
saddlebrown
sienna
brown
maroon
white
snow
honeydew
mintcream

azure
aliceblue
ghostwhite
whitesmoke
seashell
beige
oldlace
floralwhite
ivory
antiquewhite
linen
lavenderblush
mistyrose
gainsboro
lightgray
silver
darkgray
gray
dimgray
lightslategray
slategray
darkslategray
black
gray1
gray5
gray10
gray20
gray30
gray40
gray50
gray60
gray70
gray80
gray90

GLE supports these SVG/X11 standard colors (alphabetical order)

aliceblue
antiquewhite
aqua
aquamarine
azure
beige
bisque
black
blanchedalmond
blue
blueviolet
brown
burlywood
cadetblue
chartreuse
chocolate
coral
cornflowerblue
cornsilk
crimson
cyan
darkblue
darkcyan
darkgoldenrod
darkgray
darkgreen
darkkhaki
darkmagenta
darkolivegreen
darkorange
darkorchid
darkred
darksalmon
darkseagreen
darkslateblue
darkslategray
darkturquoise
darkviolet
deeppink
deepskyblue
dimgray
dodgerblue
firebrick
floralwhite
forestgreen
fuchsia
gainsboro
ghostwhite
gold
goldenrod
gray
green
greenyellow
honeydew
hotpink
indianred
indigo
ivory
khaki
lavender
lavenderblush
lawngreen
lemonchiffon
lightblue
lightcoral
lightcyan
lightgoldenrodyellow
lightgray
lightgreen
lightpink
lightsalmon
lightseagreen
lightskyblue
lightslategray
lightsteelblue
lightyellow
lime
limegreen
linen
magenta
maroon
mediumaquamarine
mediumblue
mediumorchid
mediumpurple
mediumseagreen
mediumslateblue
mediumspringgreen
mediumturquoise
mediumvioletred
midnightblue
mintcream
mistyrose
moccasin
navajowhite
navy
oldlace
olive
olivedrab
orange
orangered
orchid
palegoldenrod
palegreen
paleturquoise
palevioletred
papayawhip
peachpuff
peru
pink
plum
powderblue
purple
red
rosybrown
royalblue
saddlebrown
salmon
sandybrown
seagreen
seashell
sienna
silver
skyblue
slateblue
slategray
snow
springgreen
steelblue
tan
teal
thistle
tomato
turquoise
violet
wheat
white
whitesmoke
yellow
yellowgreen
gray1
gray5
gray10
gray20
gray30
gray40
gray50
gray60
gray70
gray80
gray90

A.9 Wall Reference



Appendix B

Index

Index