The PLPLOT Plotting Library

Tony Richardson

June 19, 1989

Contents

Chapter 1

Introduction

1.1 The PLPLOT Plotting Library

PLPLOT is a library of C functions for drawing graphs on a variety of graphics devices using the Amiga personal computer. The only assumption made about an output device is that it is capable of displaying a line, so new graphics devices may be added readily by writing a small number of device-dependent routines.

The library is Lattice C compatible and is accessible using the Lattice C linker (blink). The source code is standard C and should be relatively easy to compile for use with other compiler/linker packages.

This manual provides a description of the routines available to the programmer for producing graphical output. For most applications, the program can be device-independent, and the output can be directed to the appropriate graphics device at run-time. However, for specialized applications, where the absolute size of a graph is important, it is also possible to specify the plotting area in terms of absolute coordinates (millimeters).

Many of the underlying concepts used in the PLPLOT subroutine package are based on ideas used in Tim Pearson's PGPLOT package originally written in VAX-specific Fortran-77. Sze Tan of the University of Auckland originally developed PLPLOT on an IBM PC, and subsequently transferred it to a number of other machines. Additional features were added to allow three-dimensional plotting and better access to low-level routines. When I received PLPLOT from Sze Tan I ported it to a Data General MV/20000 and added a few features as well. I had been looking for a plotting package with the features of PLPLOT for use on the Amiga. I do not have a FORTRAN compiler, so I rewrote PLPLOT in C and ported it to the Amiga.

This document was originally written by Sze Tan. I have edited it so that it is applicable to the Amiga and the C programming language.

1.2 Installing PLPLOT

PLPLOT is distributed on a single floppy disk. The disk (labelled PLPLOT) contains the plotting library, some example programs, the source code and the documentation.

If you don't have a hard disk, then there is no installation procedure. It might be wise to make backup copies of the floppy disks though.

If you have a hard disk and want to install PLPLOT on it I recommend creating a :plplot/lib directory on your harddisk and then copying the library there. The following commands will do the trick.

```
makedir HDDISK:plplot/lib
copy plplot1:lib HDDISK:plplot/lib all
```

Be sure to substitute the name of your own hard disk for `HDDISK` in the above commands.

1.3 Linking Lattice C programs with PLPLOT

If you don't have a hard disk or didn't install the library on it, then a program can be compiled and linked with the library with the following command:

```
lc -Lm+PLPLOT:lib/plplot.lib ProgramName
```

If you have installed the PLPLOT library on a hard disk in the recommended manner, then use the following command:

```
lc -Lm+HDDISK:plplot/lib/plplot.lib ProgramName
```

Notice that the PLPLOT functions expect to be linked with the standard Lattice math library. You may want to create a new library that uses the FFP routines or provides 68881 support.

1.4 Devices Supported by PLPLOT on the Amiga

At run time the user specifies whether the graphics should be drawn on the Amiga's screen or stored in a device dependent file. If the Amiga screen is selected then a backdrop, borderless window is opened on a high resolution, interlaced screen. Two menus are available for screen and graphics control. When the window is first opened only two menu selections are available. These are the Interrupt selection under the Graphics Control menu (Interrupt can also be selected by typing CTRL-C) and the Screen to Back selection under the Screen Control menu (Screen to Back can also be selected to by holding the right Amiga key down and then hitting F). The Screen to Back selection will move the graphics screen behind any other screens you may have open. The Interrupt selection will suspend any drawing operations. After the Interrupt selection is made all other menu selections are enabled. Under the Screen Control menu the user can then select Screen to Back, Clear Screen, and Close Screen. The Screen to Back selection has already been discussed. The Clear Screen and Close Screen operations can also be selected from the keyboard by holding down the right Amiga key and typing either C or Q. The Clear Screen selection, as you've probably guessed, clears the screen. The Close Screen selection closes the screen, calls `plend` (page 21) and then calls the exit routine to end program execution. The Close Screen selection does not return control back to the user's main program. It should be used only to gracefully abort the program. Under the Graphics Control menu the only enabled menu selection is Continue (also selectable by hitting the RETURN key). This will continue program execution at the point where it was suspended. When a program uses PLPLOT to create a sequence of graphs then PLPLOT will automatically wait for the Continue selection before erasing one graph and starting the next one. After the last graph is displayed (or the first one, if only one is being drawn) PLPLOT will again wait for a Continue selection. The screen will then be closed and control returned to the user's program. This is the preferred method of closing the screen (as opposed to the Close Screen selection).

After one graph is complete, the user can move the Screen to Back to get WorkBench access and then use the GraphicDump (or similar) program to obtain hardcopy. However, if you have

a fairly high resolution printer or plotter best results will be obtained if you use a specific device driver. Three device drivers are supplied with PLPLOT. These drivers can be used as guidelines in writing your own drivers. The supplied drivers provide support for the following graphics languages:

- HP LaserJet II.
- imPRESS,
- Tektronix,

The HP LaserJet II does not actually support any type of vector graphics language. Instead a bit map is created in memory and when the graph is completed the bit map is written to a file (along with some HP specific control sequences). To keep the memory requirements (for both the Amiga and LaserJet II) low this driver only supports the 150 dot per inch mode of the laser printer. The file sizes can be rather large if multiple graphs are produced from a single main program.

Tektronix and imPRESS devices do have vector drawing support and are the simplest type devices to write drivers for. The graphics commands are written to a file on the fly, without the need for a bit map. imPRESS is a page description language for IMAGEN printers. After the graphics file is created just copy it to the appropriate device (do not send the file to prt: use either ser: or par:).

1.5 Adding Support for Other Devices

If you know the fundamentals of C programming then adding support for other devices should be relatively easy. First, become familiar with the device. Determine whether it supports direct line drawing or raster graphics. If it has a line drawing mode then use either `tektronix.c` or `impress.c` (found in the src directory) to guide you in creating the new driver. If only raster graphics (i.e. a bitmap device) are supported then use `laserjetii.c` as a guide. You will also need to add the appropriate code to `plstar.c` (also in the src directory). An examination of plstar.c should indicate what changes have to be made. A make file (LMKFILE) is provided in the root directory of the PLPLOT disk to automate the process of updating the library. You will probably want to add the name of your driver to this make file.

1.6 Organization of this Manual

The PLPLOT library has been designed so that it is easy to write programs producing graphical output without having to set up large numbers of parameters. However, more precise control of the results may be necessary, and these are accomodated by providing lower-level routines which change the system defaults. In chapter 2, the overall process of producing a graph using the high-level routines is described. Chapter 3 discusses the underlying concepts of the plotting process and introduces some of the more complex routines. The reference section of the manual, chapter 4 is an alphabetical list of the user-accessible PLPLOT functions with detailed descriptions.

Chapter 2

Simple Use of PLPLOT

2.1 Plotting a Simple Graph

We shall first consider plotting simple graphs showing the dependence of one variable upon another. Such a graph may be composed of several elements:

- A box which defines the ranges of the variables, perhaps with axes and numeric labels along its edges,
- A set of points or lines within the box showing the functional dependence,
- A set of labels for the variables and a title for the graph.

In order to draw such a graph, it is necessary to call at least four of the PLPLOT functions:

1. `plstar`, to specify the device you want to plot on,
2. `plenv`, to define the range and scale of the graph, and draw labels, axes, etc.,
3. One or more calls to `plline` or `plpoin` to draw lines or points as needed. Other more complex routines include `plbin` and `plhist` to draw histograms, `plerrx` and `plerry` to draw error-bars,
4. `plend`, to close the plot.

More than one graph can be drawn on a single set of axes by making repeated calls to the routines listed in item 3 above. The routine `plstar` needs to be called only once, unless a different output device is required.

2.2 Starting PLPLOT

Subroutine `plstar` selects a graphics device or opens a disk file to receive a plot for later display. If `plstar` is called again during a program, the previously opened file will be closed. When called, the user is prompted for a number representing the device on which the plot is to appear. The syntax for `plstar` is:

```
plstar(nx,ny);
```

`nx, ny` (int, input):

The number of plots to a page. The page is divided into `nx` × `ny` subpages, with `nx` in the horizontal direction, and `ny` in the vertical direction.

Subpages are useful for placing several graphs on a page, but all subpages are constrained to be of the same size. For greater flexibility in placing graphs on a page, read page 10 in Section 3.3 which discusses viewports.

2.3 Defining plot scales and axes

The function `plenv` is used to define the scales and axes for simple graphs. `plenv` starts a new picture on the next subpage (or a new page if necessary), and defines the ranges of the variables required. The routine will also draw a box, axes, and numeric labels if requested. The syntax for `plenv` is:

```
plenv(xmin,xmax,ymin,ymax,just,axis);
```

`xmin, xmax` (float, input):
The left and right limits for the horizontal axis.

`ymin, ymax` (float, input):
The bottom and top limits for the vertical axis.

`just` (int, input):
This should be zero or one. If `just` is one, the scales of the x-axis and y-axis will be the same (in units per millimeter); otherwise the axes are scaled independently. This parameter is useful for ensuring that objects such as circles have the correct aspect ratio in the final plot.

`axis` (int, input):
`axis` controls whether a box, tick marks, labels, axes, and/or a grid are drawn.

- `axis=-2`: No box or annotation.
- `axis=-1`: Draw box only.
- `axis= 0`: Draw box, labelled with coordinate values around edge.
- `axis= 1`: In addition to box and labels, draw the two axes X=0 and Y=0.
- `axis= 2`: As for `axis=1`, but also draw a grid at the major tick interval.
- `axis=10`: Logarithmic X axis, linear Y axis.
- `axis=11`: Logarithmic X axis, linear Y axis and draw line Y=0.
- `axis=20`: Linear X axis, logarithmic Y axis.
- `axis=21`: Linear X axis, logarithmic Y axis and draw line X=0.
- `axis=30`: Logarithmic X and Y axes.

Note: Logarithmic axes only affect the appearance of the axes and their labels, so it is up to the user to compute the logarithms prior to passing them to `plenv` and any of the other routines. Thus, if a graph has a 3-cycle logarithmic axis from 1 to 1000, we need to set `xmin`= $\log_{10} 1 = 0.0$, and `xmax`= $\log_{10} 1000 = 3.0$.

For greater control over the size of the plots, axis labelling and tick intervals, more complex graphs should make use of the functions `plvpor` (page 31), `plwind` (page 32), and `plbox` (page 18) described in Chapter 4.

2.4 Labelling the graph

The function `pllab` may be called after `plenv` to write labels on the x and y axes, and at the top of the picture. All the variables are character variables or constants. Trailing spaces are removed and the label is centred in the appropriate field. The syntax for `pllab` is:

```
pllab(xlbl,ylbl,toplbl);
```

`xlbl` (char *, input):
Pointer to string with label for the X-axis (bottom of graph).

`ylbl` (char *, input):
Pointer to string with label for the Y-axis (left of graph).

`toplbl` (char *, input):
Pointer to string with label for the plot (top of picture).

More complex labels can be drawn using the function rouplmtex. See also Section 2.5.3 on page 7 for information about the function `plptex` which writes labels within a graph.

2.5 Drawing the Graph

PLPLOT can draw graphs consisting of points with optional error bars, line segments or histograms. Functions which perform each of these actions may be called after setting up the plotting environment using `plenv`. All of the following functions draw within the box defined by `plenv`, and any lines crossing the boundary are clipped. Functions are also provided for drawing surface and contour representations of multi-dimensional functions. These are described in Chapter 3.

2.5.1 Drawing Points

`plpoin` and `plsym` mark out `n` points `(x[i],y[i])` with the specified symbol. The routines differ only in the interpretation of the symbol codes. `plpoin` uses an extended ASCII representation, with the printable ASCII codes mapping to the respective characters in the current font, and the codes from 0-31 mapping to various useful symbols. In `plsym` however, the code is a Hershey font code number. Example programs are provided which display each of the symbols available using these routines.

`plpoin(n,x,y,code);` and `plsym(n,x,y,code);`

`n` (int, input):
the number of points to plot.

`x, y` (float *, input):
pointers to arrays of the coordinates of the `n` points.

`code` (int, input):
code number of symbol to draw.

2.5.2 Drawing Lines or Curves

PLPLOT provides two functions for drawing line graphs. All lines are drawn in the currently selected color and line style. See page 12 in Section 3.7 and page 12 in Section 3.7 for information about changing these parameters.

`plline` draws a line or curve. The curve consists of `n-1` line segments joining the `n` points in the input arrays. For single line segments, `pljoin` is used to join two points.

```
plline(n,x,y);
```

`n` (int, input):
the number of points.

`x, y` (float *, input):
pointers to arrays with coordinates of the `n` points.

```
pljoin(x1,y1,x2,y2);
```

`x1, y1` (float, input):
coordinates of the first point.

`x2, y2` (float, input):
coordinates of the second point.

2.5.3 Writing Text on a Graph

`plptex` allows text to be written within the limits set by `plenv`. The reference point of a text string may be located anywhere along an imaginary horizontal line passing through the string at half the height of a capital letter. The parameter "just" specifies where along this line the reference point is located. The string is then rotated about the reference point through an angle specified by the parameters `dx` and `dy`, so that the string becomes parallel to a line joining `(x,y)` to `(x+dx,y+dy)`.

```
plptex(x,y,dx,dy,just,text);
```

`x, y` (float, input):
coordinates of the reference point.

`dx, dy` (float, input):
these specify the angle at which the text is to be printed. The text is written parallel to a line joining the points `(x,y)` to `(x+dx,y+dy)` on the graph.

`just` (float, input):
determines justification of the string by specifying which point within the string is placed at the reference point `(x,y)`. This parameter is a fraction of the distance along the string. Thus if `just=0.0`, the reference point is at the left-hand edge of the string. If `just=0.5`, it is at the center and if `just=1.0`, it is at the right-hand edge.

`text` (char *, input):
pointer to the string of characters to be written.

2.5.4 More Complex Graphs (Histograms and Error Bars)

Functions rouplbin and `plhist` (page 24) are provided for drawing histograms, and functions rouplerrx and `plerry` (page 23) draw error bars about specified points. They are described in detail in Chapter 4.

2.6 Finishing up

Before the end of the program, always call `plend` (page 21) to close any output plot file and to free up any memory that may have been allocated. For devices that have separate graphics and text modes, `plend` always resets the device into text mode. If it is required to switch between modes within a program, the functions `plgra` (page 23) and `pltext` (page 31) set the device to graphics and text modes respectively.

Chapter 3

Advanced Use of PLPLOT

In this chapter, we describe more precisely how to control the position and scaling of a graph, how to alter the low-level line and character attributes, and how to use the functions in PLPLOT for drawing three-dimensional surface plots and contour plots.

3.1 Basic PLPLOT Concepts

When drawing a graph, the programmer usually wishes to specify the coordinates of the points to be plotted in terms of the values of the variables involved. These coordinates are called *world coordinates*, and may have any floating-point value representable by the computer. The *window* refers to the rectangular region of world-coordinate space which is to be graphed. This window is mapped onto a rectangular region of the *view surface*, which is (a portion) of the screen or sheet of paper in the output device. This physical region onto which the window is mapped is called the *viewport*. Before a graph can be drawn, the program must define both the window and the viewport by calling appropriate routines in PLPLOT.

3.2 Specifying the View Surface

The first thing that a graphics program must do is to tell PLPLOT which device it is going to use, and how this device is to be divided up for graph plotting. There are two routines that do this, `plstar` (page 29), which prompts at the console for the output device type and `plbeg` (page 17), which expects to be supplied the code number of the device as an argument. The code numbers required by `plbeg` are the same as displayed by `plstar` when it prompts for a device.

Besides selecting the device, `plstar` and `plbeg` allow the user to divide the output device plotting area into several subpages of equal size, each of which can be used separately. The routine `pladv` is used to advance to a particular subpage or to the next subpage. The screen is cleared (after waiting for the user to select Continue from the Graphics Control menu or hit the RETURN key), or a new piece of paper is loaded, if a new subpage is requested when there are no subpages left on the current page. When a page is divided into subpages, the default character, symbol and tick sizes are scaled inversely as the square root of the number of subpages in the vertical direction.

At the end of a plotting program, it is important to close the plotting device by calling `plend` (page 21). This flushes any internal buffers and frees any memory that may have been allocated.

Note that if `plstar` or `plbeg` is called more than once during a program to change the output device, an automatic call to `plend` is made before the new device is opened.

3.3 Defining the Viewport

After defining the view surface, it is necessary to define the portion of this surface which is to be used for plotting the graph. All lines and symbols (except for labels drawn by `plbox`, `plmtex` and `pllab`) are clipped at the viewport boundaries.

Viewports are created within the current subpage. If the division of the output device into equally sized subpages is inappropriate, it is best to specify only a single subpage which occupies the entire output device (by setting `nx=1` and `ny=1` in `plbeg` or `plstar`), and use one of the viewport specification subroutines below to place the plot in the desired position on the page.

There are two methods for specifying the viewport size, using the subroutines `plvpor` (page 31) and `plsvpa` (page 30). Each of these has the format:

```
plvpor(xmin,xmax,ymin,ymax);
plsvpa(xmin,xmax,ymin,ymax);
```

where in the case of `plvpor`, the arguments are given in *normalized subpage coordinates* which are defined to run from 0.0 to 1.0 along each edge of the subpage. Thus for example,

```
plvpor(0.0,0.5,0.5,1.0);
```

uses the top left quarter of the current subpage.

In order to get a graph of known physical size, the routine `plsvpa` defines the viewport in terms of absolute coordinates (millimetres) measured from the bottom left-hand corner of the current subpage. This routine should only be used when the size of the view surface is known, and a definite scaling is required.

To help the user call `plsvpa` correctly, the routine `plgspa` (page 24) is provided which returns the positions of the extremities of the current subpage measured in millimetres from the bottom left-hand corner of the device. Thus, if to set up a viewport with a 10.0 mm margin around it within the current subpage, the following sequence of calls may be used

```
plgspa(xmin,xmax,ymin,ymax);
plsvpa(10.0,xmax-xmin-10.0,10.0,ymax-ymin-10.0);
```

A further routine `plvsta` (page 31) is available which sets up a standard viewport within the current subpage with suitable margins on each side of the viewport. This may be used for simple graphs, as it leaves enough room for axis labels and a title. This standard viewport is that used by `plenv` (see Section 3.6).

3.4 Defining the Window

The window must be defined after the viewport in order to map the world coordinate rectangle into the viewport rectangle. The routine `plwind` (page 32) is used to specify the rectangle in world-coordinate space. For example, if we wish to plot a graph showing the collector current I_C as a function of the collector to emitter voltage V_{CE} for a transistor where $0 \leq I_C \leq 10.0\,\mathrm{mA}$ and $0 \leq V_{CE} \leq 12.0\,\mathrm{V}$, we would call the function `plwind` as follows:

```
plwind(0.0,12.0,0.0,10.0);
```

Note that each of the arguments is a floating point number, and so the decimal points are required. If the order of either the X limits or Y limits is reversed, the corresponding axis will point in the opposite sense, (i.e., right to left for X and top to bottom for Y). The window must be defined before any calls to the routines which actually draw the data points. Note however that `plwind` may also be called to change the window at any time. This will affect the appearance of objects drawn later in the program, and is useful for drawing two or more graphs with different axes on the same piece of paper.

3.5 Anotating the Viewport

The routine `plbox` (page 18) is used to specify whether a frame is drawn around the viewport and to control the positions of the axis subdivisions and numeric labels. For our simple graph of the transistor characteristics, we may wish to draw a frame consisting of lines on all four sides of the viewport, and to place numeric labels along the bottom and left hand side. We can also tell PLPLOT to choose a suitable tick interval and the number of subticks between the major divisions based upon the data range specified to `plwind`. This is done using the following statement

```
plbox("BCNST",0.0,0,"BCNSTV",0.0,0);
```

Another routine `pllab` (page 25) provides for text labels for the bottom, left hand side and top of the viewport. These labels are not clipped, even though they lie outside the viewport (but they are clipped at the subpage boundaries). `pllab` actually calls the more general routine `plmtex` which can be used for plotting labels at any point relative to the viewport. For our example, we may use

```
pllab("V\\dCE\\u (Volts)","I\\dC\\u (mA)","TRANSISTOR CHARACTERISTICS");
```

Note that `\\d` and `\\u` are escape sequences (see page 13) which allow subscripts and superscripts to be used in text. They are described more fully later in this chapter.

The heights of characters used for the axis and graph labels can be changed by means of the routine `plschr` (page 28).

3.6 The routine `plenv`

Having to call `pladv`, `plvpor`, `plwind` and `plbox` is excessively cumbersome for drawing simple graphs. Subroutine `plenv` combines all four of these in one subroutine, using the standard viewport, and a limited subset of the capabilities of `plbox`. For example, the graph described above could be initiated by the call:

```
plenv(0.0,12.0,0.0,10.0,0,0);
```

which is equivalent to the following series of calls:

```
pladv(0);
plvsta();
plwind(0.0,12.0,0.0,10.0);
plbox("BCNST",0.0,0,"BCNSTV",0.0,0);
```

3.7 Setting Line Attributes

The graph drawing routines may be freely mixed with those described in this section which allow the user to control line colour and styles. The attributes set up by these routines apply modally, i.e, all subsequent objects (lines, characters and symbols) plotted until the next change in attributes are affected in the same way. The only exception to this rule is that characters and symbols are not affected by a change in the line style, but are always drawn using a continuous line.

Line colour is set using the routine `plcol` (page 21). The argument is ignored for devices which can only plot in one colour but some terminals support line erasure by plotting in colour zero. For HP plotters, these colours map to the various pens.

Line style is set using the routine `plstyl` (page 30). A broken line is specified in terms of a repeated pattern consisting of marks (pen down) and spaces (pen up). The arguments to this routine consist of the number of elements in the line, followed by two pointers to integer arrays specifying the mark and space lengths in micrometres. Thus a line consisting of long and short dashes of lengths 4 mm and 2 mm, separated by spaces of length 1.5 mm is specified by:

```
MARK[0]=4000;
MARK[1]=2000;
SPACE[0]=1500;
SPACE[1]=1500;
plstyl(2,MARK,SPACE);
```

To return to a continuous line, just call `plstyl` with first argument set to zero.

3.8 Setting Character and Symbol Attributes

The routine `plfont` (page 23) sets up the default font for all character strings. It may be over-ridden for (a portion) of a string by using an escape sequence within the text, as described below. Four fonts are available, the default font (1) is simple and fastest to draw, while the others are useful for presentation plots on a high-resolution device.

The font codes are interpreted as follows:

- `font = 1`: normal simple font
- `font = 2`: roman font
- `font = 3`: italic font
- `font = 4`: script font

The routine `plschr` (page 28) is used to set up the size of subsquent characters drawn. The actual height of a character is the product of the default character size and a scaling factor. If no call is made to `plschr`, the default character size is set up depending on the number of subpages defined in the call to `plstar` or `plbeg`, and the scale is set to 1.0. Under normal circumstances, it is recommended that the user does not alter the default height, but simply uses the scale parameter. This can be done by calling `plschr` with `def=0.0` and `scale` set to the desired multiple of the default height. If the default height is to be changed, `def` is set to the new default height in millimetres, and the new character height is again set to `def` multiplied by `scale`.

The routine `plssym` (page 29) sets up the size of all subsequent symbols drawn by calls to `plpoin` and `plsym`. It operates analogously to `plschr` as described above.

The lengths of major and minor ticks on the axes are set up by the routines `plsmaj` (page 29) and `plsmin` (page 29).

3.9 Escape sequences in text

The routines which draw text all allow you to include escape sequeces in the text to be plotted. These are character sequences which are are interpreted as instructions to change font, draw superscripts and subscripts, draw non-ASCII (e.g., Greek letters) etc. All escape sequences start with a double backslash character (`\\`).

The following escape sequences are defined:

- `\\u`: move up to the superscript position (ended with `\d`)
- `\\d`: move down to subscript position (ended with `\\u`)
- `\\b`: backspace (to allow overprinting)
- `\\\\`: backslash
- `\\+`: toggle overline mode
- `\\-`: toggle underline mode
- `\\gx`: Greek letter corresponding to Roman letter `x` (see below)
- `\\fn`: switch to normal font
- `\\fr`: switch to Roman font
- `\\fi`: switch to italic font
- `\\fs`: switch to script font
- `\\(nnn)`: Hershey character nnn (1 to 4 decimal digits)

Sections of text can have an underline or overline appended. For example, the string

$$\overline{\mathrm{S}}(\underline{\mathrm{freq}})$$

is obtained by specifying `"\\+S\\+(\\-freq\\-)"`.

Greek letters are obtained by `\\g` followed by a Roman letter. Table 3.1 shows how these letters map into Greek characters.

3.10 Three dimensional surface plots

PLPLOT includes routines that will represent a single-valued function of two variables as a surface. In this section, we shall assume that the function to be plotted is `Z[X][Y]`, where `Z` represents the dependent variable and `X` and `Y` represent the independent variables.

As usual, we would like to refer to a three dimensional point `(X,Y,Z)` in terms of some meaningful user-specified coordinate system. These are called *three-dimensional world coordinates*. We need to specify the ranges of these coordinates, so that the entire surface is contained within

Roman	A	B	G	D	E	Z	Y	H	I	K	L	M
Greek	A	B	Γ	Δ	E	Z	H	Θ	I	K	Λ	M
Roman	N	C	O	P	R	S	T	U	F	X	Q	W
Greek	N	Ξ	O	Π	P	Σ	T	Υ	Φ	X	Ψ	Ω
Roman	a	b	g	d	e	z	y	h	i	k	l	m
Greek	α	β	γ	δ	ϵ	ζ	η	θ	ι	κ	λ	μ
Roman	n	c	o	p	r	s	t	u	f	x	q	w
Greek	ν	ξ	o	π	ρ	σ	τ	υ	ϕ	χ	ψ	ω

Table 3.1: Roman characters corresponding to Greek characters

the cuboid defined by `xmin` $<$ `x` $<$ `xmax`, `ymin` $<$ `y` $<$ `ymax` and `zmin` $<$ `z` $<$ `zmax`. Typically, we shall want to view the surface from a variety of angles, and to facilitate this, a two-stage mapping of the enclosing cuboid is performed. Firstly, it is mapped into another cuboid called the *normalized box* whose size must also be specified by the user, and secondly this normalized box is viewed from a particular azimuth and elevation so that it can be projected onto the two-dimensional window.

This two-stage transformation process allows considerable flexibility in specifying how the surface is depicted. The lengths of the sides of the normalized box are independent of the world coordinate ranges of each of the variables, making it possible to use "reasonable" viewing angles even if the ranges of the world coordinates on the axes are very different. The size of the normalized box is determined essentially by the size of the two-dimensional window into which it is to be mapped. The normalized box is centred about the origin in the x and y directions, but rests on the plane `z` $= 0$. It is viewed by an observer located at altitude `alt` and azimuth `az`, where both angles are measured in degrees. The altitude should be restricted to the range zero to ninety degrees for proper operation, and represents the viewing angle above the xy plane. The azimuth is defined so that when `az=0`, the observer sees the xz plane face on, and as the angle is increased, the observer moves clockwise around the box as viewed from above the xy plane. The azimuth can take on any value.

The first step in drawing a surface plot is to decide on the size of the two-dimensional window and the normalized box. For example, we could choose the normalized box to have sides of length

```
basex=2.0;
basey=4.0;
height=3.0;
```

A reasonable range for the x coordinate of the two-dimensional window is -2.5 to +2.5, since the length of the diagonal across the base of the normalized box is $\sqrt{2^2+4^2} = 2\sqrt{5}$, which fits into this coordinate range. A reasonable range for the y coordinate of the two dimensional window in this case is -2.5 to +4, as the the projection of the normalized box lies in this range for the allowed range of viewing angles.

The routine `plwind` (page 32) or `plenv` (page 22) is used in the usual way to establish the size of the two-dimensional window. The routine `plw3d` (page 31) must then be called to establish the range of the three dimensional world coordinates, the size of the normalized box and the viewing angles. After calling `plw3d`, the actual surface is drawn by a call to `plot3d` (page 26).

For example, if the three-dimensional world-coordinate ranges are $-10.0 \leq$ `x` ≤ 10.0, $-3.0 \leq$ `y` $\leq +7.0$ and $0.0 \leq$ `z` ≤ 8.0, we could use the following statements:

```
xmin2d = -2.5;
```

```
xmax2d =  2.5;
ymin2d = -2.5;
ymax2d =  4.0;
plenv(xmin2d,xmax2d,ymin2d,ymax2d);
basex = 2.0;
basey = 4.0;
height = 3.0;
xmin = -10.0;
xmax = 10.0;
ymin = -3.0;
ymax = 7.0;
zmin = 0.0;
zmax = 8.0;
alt = 45.0;
az = 30.0;
plw3d(basex,basey,height,xmin,xmax,ymin,ymax,zmin,zmax,alt,az);
plot3d(x,y,z,work,ly,nx,ny,opt);
```

The values of the function are stored in a two-dimensional array `z[][ly]` where the array element `z[i][j]` contains the value of the function at the point x_i, y_j. Note that the values of the independent variables x_i and y_j do not need to be equally spaced, but they must lie on a rectangular grid. Thus two further arrays `x[nx]` and `y[ny]` are required as arguments to `plot3d` to specify the values of the independent variables. The values in the arrays x and y must be strictly increasing with the index. The argument `opt` specifies how the surface is outlined. If `opt=1`, a line is drawn representing z as a function of x for each value of y, if `opt=2`, a line is drawn representing z as a function of y for each value of x, and if `opt=3`, a net of lines is drawn. The first two options may be preferable if one of the independent variables is to be regarded as a parameter, whilst the third is better for getting an overall picture of the surface. The final parameter `work` is an integer array with at least four times `max(nx,ny)` points, which is required for workspace.

Labelling a three-dimensional plot is somewhat more complicated than a two dimensional plot due to the need for skewing the characters in the label so that they are parallel to the coordinate axes. The routine `plbox3` (page 19) thus combines the functions of box drawing and labelling. Its parameters are described more fully in Chapter 4.

The routine `plside3` (page 28) can be used to draw sides on three-dimensional plots to give them a more solid look. Grid lines perpendicular to the z axis can be drawn with the routine `plgrid3` (page 23). Both routines are decribed in Chapter 4.

3.11 Contour Plots

A routine is available in PLPLOT which performs a contour plot of data stored in a two-dimensional array. A contour following algorithm is used, so that it is possible to use non-continuous line styles.

The routine `plcont` (page 21) has the form

```
plcont(z,nx,ny,kx,lx,ky,ly,clevel,nlevel,tr)
```

where `z` is the two-dimensional array of size `nx` × `ny` containing samples of the function to be contoured. The parameters `kx`, `lx`, `ky` and `ly` specify the portion of `z` that is to be considered. The array `clevel` of length `nlevel` is a list of the desired contour levels.

The path of each contour is initially computed in terms of the values of the array indicies which range from 1 to `nx` in the first index and from 1 to `ny` in the second index. Before these can be drawn in the current window (see page 10 in Section 3.4), it is necessary to convert from these array indicies into world coordinates. This is done by passing a pointer to a user-defined function to `plcont`. This function pointer is the last argument `tr`. This function must be declared as type `void` in the module which calls `plcont`. This transformation function must have the parameter list

```
void tr(x,y,tx,ty);
```

where `(x,y)` is the point through which the contour runs expressed in terms of array indicies, and `(tx,ty)` are pointers to float variables which are the world coordinates of the point which corresponds to these indicies.

Often, the transformation between array indicies and world coordinates can be expressed as a linear transformation. A routine is provided within the library which can be passed to `plcont` as the parameter `tr`. This transformation routine is as follows:

```
#include "plplot.h"

void xform(x,y,tx,ty)
float x, y, *tx, *ty;
{
      extern float tr[];

      *tx = tr[0]*x + tr[1]*y + tr[2];
      *ty = tr[3]*x + tr[4]*y + tr[5];
}
```

Thus by setting up the values in the array `tr[]`, we can apply an arbitrary translation, rotation and/or shear to the array before drawing out the contours. By defining other transformation subroutines, it is possible to draw contours wrapped around polar grids etc.

As an example in setting up `tr[]`, suppose that the array `z` is of size 21×41 and contains the values of the function $z[x][y]$, where x ranges from 0.0 to 4.0 and y ranges from -8.0 to 8.0. Furthermore, let us also suppose that the window (as defined using `plenv` or `plwind`) covers this range of world coordinates. Since we wish the index (1,1) in array `z` to map to (0.0,-8.0) and the index (21,41) to map to (4.0,8.0), and for there to be no skew in the axes, we should choose elements of `tr[]` so that

$$tx = 0.2x - 0.2$$
$$ty = 0.4y - 8.4$$

and so `tr[0]=0.2`, `tr[1]=0.0`, `tr[2]=-0.2`, `tr[3]=0.0`, `tr[4]=0.4`, `tr[5]=-8.4`.

Chapter 4

PLPLOT Subroutine Reference

All the PLPLOT subroutines are listed below in alphabetical order.

`pladv(sub);`

Advances to the next subpage if `sub=0` or to the specified subpage if `sub` is non-zero. This routine is called automatically (with `sub=0`) by `plenv`, but if `plenv` is not used, `pladv` must be called after `plstar` but before defining the viewport.

`sub` (int, input):
Specifies the subpage number (starting from 1 in the top left corner and increasing along the rows) to which to advance. Set to zero to advance to the next subpage.

`plbeg(dev,nx,ny);`

Alternative to `plstar` for initializing the plotting package. The device number of the desired output device must be supplied as an argument. The mapping from device numbers to devices varies from one implementation to another, and so use of `plstar` which prompts for the device type is recommended. The device codes are the same as those printed out by `plstar`. This routine also divides the output device into `nx` by `ny` sub-pages, each of which may be used independently. The subroutine `pladv` is used to advance from one subpage to the next.

`dev` (int, input):
Device number of the required output device.

`nx` (int, input):
Number of subpages to divide output page in the horizontal direction.

`ny` (int, input):
Number of subpages to divide output page in the vertical direction.

`plbin(nbin,x,y,cen);`

Plots a histogram consisting of `n` bins. The value associated with the `i`'th bin is placed in `x[i]`, and the number of points in the bin is placed in `y[i]`. For proper operation, the values in `x[i]` must form a strictly increasing sequence. If `centre` is false, `x[i]` is the left-hand edge of the `i'th` bin, and if `centre` is true, the bin boundaries are placed midway between the values in the `x` array. Also see `plhist` for drawing histograms from unbinned data.

`nbin` (int, input):
Number of bins (i.e., number of values in x and y arrays.)

`x` (float *, input):
Pointer to array containing values associated with bins. These must form a strictly increasing sequence.

`y` (float *, input):
Pointer to array containing number of points in bin. This is a **float** array so as to allow histograms of probabilities, etc.

`cen` (int, input):
Indicates whether the values in `x` represent the lower bin boundaries (`cen=0`) or whether the bin boundaries are to be midway between the `x` values (`cen=1`). If the values in `x` are equally spaced and `cen=1`, the values in `x` are the centre values of the bins.

```
plbox(xopt,xtick,nxsub,yopt,ytick,nysub);
```

Draws a box around the currently defined viewport, and labels it with world coordinate values appropriate to the window. Thus `plbox` should only be called after defining both viewport and window. The character strings `xopt` and `yopt` specify how the box should be drawn as described below. If ticks and/or subticks are to be drawn for a particular axis, the tick intervals and number of subintervals may be specified explicitly, or they may be defaulted by setting the appropriate arguments to zero.

`xopt` (char *, input):
Pointer to character string specifying options for horizontal axis. The string can include any combination of the following letters (upper or lower case) in any order:

- `a` : Draws axis, X-axis is horizontal line `y=0`, and Y-axis is vertical line `y=0`.
- `b` : Draws bottom (X) or left (Y) edge of frame.
- `c` : Draws top (X) or right (Y) edge of frame.
- `g` : Draws a grid at the major tick interval.
- `i` : Inverts tick marks, so they are drawn outwards, rather than inwards.
- `l` : Labels axis logarithmically. This only affects the labels, not the data, and so it is necessary to compute the logarithms of data points before passing them to any of the drawing routines.
- `m` : Writes numeric labels at major tick intervals in the unconventional location (above box for X, right of box for Y).
- `n` : Writes numeric labels at major tick intervals in the conventional location (below box for X, left of box for Y).
- `s` : Enables subticks between major ticks, only valid if `t` is also specified.
- `t` : Draws major ticks.

`xtick` (float, input):
World coordinate interval between major ticks on the x axis. If it is set to zero, PLPLOT automatically generates a suitable tick interval

`nxsub` (int, input):
Number of subintervals between major x axis ticks for minor ticks. If it is set to zero, PLPLOT automatically generates a suitable minor tick interval.

`yopt` (char *, input):
Pointer to character string specifying options for vertical axis. The string can include any combination of the letters defined above for `xopt`, and in addition may contain:

- `v` : Write numeric labels for vertical axis parallel to the base of the graph, rather than parallel to the axis.

`ytick` (real, input):
World coordinate interval between major ticks on the y axis. If it is set to zero, PLPLOT automatically generates a suitable tick interval

`nysub` (int, input):
Number of subintervals between major y axis ticks for minor ticks. If it is set to zero, PLPLOT automatically generates a suitable minor tick interval.

```
plbox3(xopt,xlabel,xtick,nxsub,yopt,ylabel,ytick,nysub,
       zopt,zlabel,ztick,nzsub);
```

Draws axes, numeric and text labels for a three-dimensional surface plot. See Section 3.10 on page 13 for a more complete description of three-dimensional plotting.

`xopt` (char *, input):
Pointer to character string specifying options for the x axis. The string can include any combination of the following letters (upper or lower case) in any order:

- `b` : Draws axis at base, at height `z=zmin` where `zmin` is defined by call to `plw3d`. This character must be specified in order to use any of the other options.
- `i` : Inverts tick marks, so they are drawn downwards, rather than upwards.
- `l` : Labels axis logarithmically. This only affects the labels, not the data, and so it is necessary to compute the logarithms of data points before passing them to any of the drawing routines.
- `n` : Writes numeric labels at major tick intervals.
- `s` : Enables subticks between major ticks, only valid if `t` is also specified.
- `t` : Draws major ticks.
- `u` : If this is specified, the text label for the axis is written under the axis.

`xlabel` (char *, input):
Pointer to character string specifying text label for the x axis. It is only drawn if `u` is in the `xopt` string.

`xtick` (float, input):
World coordinate interval between major ticks on the x axis. If it is set to zero, PLPLOT automatically generates a suitable tick interval

`nxsub` (int, input):

Number of subintervals between major x axis ticks for minor ticks. If it is set to zero, PLPLOT automatically generates a suitable minor tick interval.

`yopt` (char *, input):
Pointer to character string specifying options for the y axis. The string is interpreted in the same way as `xopt`.

`ylabel` (char *, input):
Pointer to character string specifying text label for the y axis. It is only drawn if `u` is in the `yopt` string.

`ytick` (float, input):
World coordinate interval between major ticks on the y axis. If it is set to zero, PLPLOT automatically generates a suitable tick interval

`nysub` (int, input):
Number of subintervals between major y axis ticks for minor ticks. If it is set to zero, PLPLOT automatically generates a suitable minor tick interval.

`zopt` (char *, input):
Pointer to character string specifying options for the z axis. The string can include any combination of the following letters (upper or lower case) in any order:

- `b` : Draws z axis to the left of the surface plot.
- `c` : Draws z axis to the right of the surface plot.
- `i` : Inverts tick marks, so they are drawn away from the centre.
- `l` : Labels axis logarithmically. This only affects the labels, not the data, and so it is necessary to compute the logarithms of data points before passing them to any of the drawing routines.
- `m` : Writes numeric labels at major tick intervals on the right-hand verical axis.
- `n` : Writes numeric labels at major tick intervals on the left-hand verical axis.
- `s` : Enables subticks between major ticks, only valid if `t` is also specified.
- `t` : Draws major ticks.
- `u` : If this is specified, the text label is written beside the left-hand axis.
- `v` : If this is specified, the text label is written beside the right-hand axis.

`zlabel` (char *, input):
Pointer to character string specifying text label for the z axis. It is only drawn if `u` or `v` are in the `zopt` string.

`ztick` (float, input):
World coordinate interval between major ticks on the z axis. If it is set to zero, PLPLOT automatically generates a suitable tick interval

`nzsub` (int, input):
Number of subintervals between major z axis ticks for minor ticks. If it is set to zero, PLPLOT automatically generates a suitable minor tick interval.

`plclr`

Clears the graphics screen of an interactive device, or ejects a page on a plotter.

```
plcol(colour);
```

Sets the colour for subsequent lines.

`colour` (int, input):
Integer representing the colour.

```
plcont(z,nx,ny,kx,lx,ky,ly,clevel,nlevel,tr);
```

Draws a contour plot of the data in `z[nx][ny]`, using the `nlevel` contour levels specified by `clevel`. Only the region of the array from `kx` to `lx` and from `ky` to `ly` is plotted out. A transformation routine `tr` is used to map indicies within the array to the world coordinates (see Section 3.11 on page 3.11).

`z` (float *, input):
Pointer to a two-dimensional array containing data to be contoured.

`nx, ny` (int, input):
Physical dimensions of array `z`.

`kx, lx` (int, input):
Range of `x` indicies to consider.

`ky, ly` (int, input):
Range of `y` indicies to consider.

`clevel` (float *, input):
Pointer to array specifying levels at which to draw contours

`nlevel` (int, input):
Number of contour levels to draw

`tr` (void *,input):
Pointer to function that defines transformation between indicies in array `z` and the world coordinates. The function should have the form

```
tr(x,y,tx,ty);
```

`x, y` (float, input):
Specifies the position in the array through which the contour runs in terms of the array indicies

`tx, ty` (float *, output):
Pointers to the world coordinates corresponding to the point `(x,y)`

A transformation function `xform` is provided for simple linear mappings. See Section 3.11 for information.

```
plend();
```

Ends a plotting session, tidies up all the output files, switches interactive devices back into text mode and frees up any memory that was allocated. Must be called before end of program.

```
plenv(xmin,xmax,ymin,ymax,just,axis);
```

Sets up plotter environment for simple graphs by calling `pladv` and setting up viewport and window to sensible default values. `plenv` leaves enough room around most graphs for axis labels and a title. When these defaults are not suitable, use the individual routines `plvspa` or `PLVPOR` for setting up the viewport, `plwind` for defining the window, and `PLBOX` for drawing the box.

`xmin` (float, input):
Value of x at left-hand edge of window.

`xmax` (float, input):
Value of x at right-hand edge of window.

`ymin` (float, input):
Value of y at bottom edge of window.

`ymax` (float, input):
Value of y at top edge of window.

`just` (int, input):
If `just=0`, the x and y axes are scaled independently to use as much of the screen as possible, but if `just=1`, the scales of the x and y axes are made equal.

`axis` (int, input):
Controls drawing of the box around the plot:

- -2: No box or annotation.
- -1: Draw box only.
- 0: Draw box, labelled with coordinate values around edge.
- 1: In addition to box and labels, draw the two axes `x=0` and `y=0`.
- 2: As for `axis=1`, but also draw a grid at the major tick interval.
- 10: Logarithmic x axis, linear y axis.
- 11: Logarithmic x axis, linear y axis and draw line `y=0`.
- 20: Linear x axis, logarithmic y axis.
- 21: Linear x axis, logarithmic y axis and draw line `x=0`.
- 30: Logarithmic x and y axes.

```
plerrx(n,xmin,xmax,y)
```

Draws a set of `n` horizontal error bars, the `i`'th error bar extending from `xmin[i]` to `xmax[i]` at y coordinate `y[i]`. The terminals of the error bar are of length equal to the minor tick length (settable using `plsmin`).

`n` (int, input):

Number of error bars to draw.

`xmin` (float *, input):
Pointer to array with x coordinates of left-hand endpoint of error bars.

`xmax` (float *, input):
Pointer to array with x coordinates of right-hand endpoint of error bars.

`y` (float *, input):
Pointer to array with y coordinates of error bar.

```
plerry(n,x,ymin,ymax);
```

Draws a set of `n` vertical error bars, the `i`'th error bar extending from `ymin[i]` to `ymax[i]` at x coordinate `x[i]`. The terminals of the error bar are of length equal to the minor tick length (settable using `plsmin`).

`n` (int, input):
Number of error bars to draw.

`x` (float *, input):
Pointer to array with x coordinates of error bars.

`ymin` (float *, input):
Pointer to array with y coordinates of lower endpoint of error bars.

`ymax` (float *, input):
Pointer to array with y coordinate of upper endpoint of error bar.

```
plfont(font);
```

Sets the default character font for subsequent character drawing. Also affects symbols produced by `plpoin`.

`font` (int, input):
Specifies the font:

- 1: Normal font (simplest and fastest)
- 2: Roman font
- 3: Italic font
- 4: Script font

```
plgra();
```

Sets an interactive device to graphics mode, used in conjunction with `pltext` to allow graphics and text to be interspersed.

```
plgrid3(ztick);
```

Draws grid lines perpendicular to the z axis. For use only when doing three dimensional plots. If used this routine must be called after calling `plot3d` first (in order for the hidden line removal to work).

`ztick` (float, input):
World coordinate interval between grid lines on the z axis. Usually this variable has the same value as `ztick` in `plbox3`. If `ztick` is zero PLPLOT will automatically generate a suitable tick interval.

```
plgspa(xmin,xmax,ymin,ymax);
```

Gets the size of the current subpage in millimetres measured from the bottom left hand corner of the output device page or screen. Can be used in conjunction with `plsvpa` for setting the size of a viewport in absolute coordinates (millimetres).

`xmin` (float *, output):
Pointer to variable with position of left hand edge of subpage in millimetres

`xmax` (float *, output):
Pointer to variable with position of right hand edge of subpage in millimetres

`ymin` (float *, output):
Pointer to variable with position of bottom edge of subpage in millimetres

`ymax` (float *, output):
Pointer to variable with position of top edge of subpage in millimetres

```
plhist(n,data,datmin,datmax,nbin,oldwin);
```

Plots a histogram from `n` data points stored in the array `data`. This routine bins the data into `nbin` bins equally spaced between `datmin` and `datmax`, and calls `plbin` to draw the resulting histogram. Parameter `oldwin` allows the histogram either to be plotted in an existing window or causes `plhist` to call `plenv` with suitable limits before plotting the histogram.

`n` (int, input):
Number of data points

`data` (float *, input):
Pointer to array with values of the `n` data points.

`datmin` (float, input):
Left-hand edge of lowest-valued bin.

`datmax` (float, input):
Right-hand edge of highest-valued bin.

`nbin` (int, input):
Number of (equal-sized) bins into which to divide the interval `xmin` to `xmax`

`oldwin` (int, input):

If one, the histogram is plotted in the currently-defined window, and if zero, `plenv` is called automatically before plotting.

```
pljoin(x1,y1,x2,y2);
```

Joins the point `(x1,y1)` to `(x2,y2)`.

`x1` (float, input):
x coordinate of first point.

`y1` (float, input):
y coordinate of first point.

`x2` (float, input):
x coordinate of second point.

`y2` (float, input):
y coordinate of second point.

```
pllab(xlabel,ylabel,tlabel);
```

Routine for writing simple labels. Use `plmtex` for more complex labels.

`xlabel` (char *, input):
Label for horizontal axis.

`ylabel` (char *, input):
Label for vertical axis.

`tlabel` (char *, input):
Title of graph.

```
plline(n,x,y);
```

Draws `n-1` line segments joining points `(x[i],y[i])`.

`n` (int, input):
Number of points to join.

`x` (float *, input):
Pointer to array with x coordinates of points.

`y` (float *, input):
Pointer to array with y coordinates of points.

```
plmtex(side,disp,pos,just,text);
```

Writes text at a specified position relative to the viewport boundaries. Text may be written inside or outside the viewport, but is clipped at the subpage boundaries. The reference point of a string lies along a line passing through the string at half the height of a capital letter. The position of the reference point along this line is determined by `just`, and the position of the reference point relative to the viewport is set by `disp` and `pos`.

`side` (char *, input):
Specifies the side of the viewport along which the text is to be written. The string must be one of:

- b: Bottom of viwport.
- l: Left of viewport, text written parallel to edge.
- lv: Left of viewport, text written at right angles to edge.
- r: Right of viewport, text written parallel to edge.
- rv: Right of viewport, text written at right angles to edge.
- t: Top of viewport.

`disp` (float, input):
Position of the reference point of string, measured outwards from the specified viewport edge in units of the current character height. Use negative `disp` to write within the viewport.

`pos` (float, input):
Position of the reference point of string along the specified edge, expressed as a fraction of the length of the edge.

`just` (float, input):
Specifies the position of the string relative to its reference point. If `just=0`, the reference point is at the left and if `just=1`, it is at the right of the string. Other values of `just` give intermediate justifications.

`text` (char *, input):
The string to be written out.

```
plot3d(x,y,z,work,ly,nx,ny,opt);
```

Plots a three dimensional surface plot within the environment set up by `plw3d`. The surface is defined by the two-dimensional array `z[][ly]`, the point `z[i][j]` being the value of the function at `(x[i],y[j])`. Note that the points in arrays `x` and `y` do not need to be equally spaced, but must be stored in ascending order. The parameter `opt` controls the way in which the surface is displayed. See Section 3.10 on page 13 for further details.

`x` (float *, input):
Pointer to set of x coordinate values at which the function is evaluated.

`y` (float *, input):
Pointer to set of y coordinate values at which the function is evaluated.

`z` (float *, input):
Pointer to two dimensional array with set of function values.

`work` (int *, input and output):
Pointer to work array of dimension at least four times the maximum of `nx` and `ny`.

`ly` (int, input):

Declared second dimension of `z` array.

`nx` (int, input):
Number of `x` values at which function is evaluated.

`ny` (int, input):
Number of `y` values at which function is evaluated.

`opt` (int, input):
Determines the way in which the surface is represented:

- 1: Lines are drawn showing `z` as a function of `x` for each value of `y[j]`.
- 2: Lines are drawn showing `z` as a function of `y` for each value of `x[i]`.
- 3: Network of lines is drawn connecting points at which function is defined.

```
plpoin(n,x,y,code);
```

Marks out a set of `n` points at positions `(x(i),y(i))`, using the symbol defined by `code`. If `code` is between 32 and 127, the symbol is simply the printable ASCII character in the default font.

`n` (int, input):
Number of points to be marked.

`x` (float *, input):
Pointer to array with set of x coordinate values for the points.

`y` (float *, input):
Pointer to array with set of y coordinate values for the points.

`code` (int, input):
Code number for the symbol to be plotted.

```
plptex(x,y,dx,dy,just,text);
```

Writes text at a specified position and inclination within the viewport. Text is clipped at the viewport boundaries. The reference point of a string lies along a line passing through the string at half the height of a capital letter. The position of the reference point along this line is determined by `just`, the reference point is placed at world coordinates `(x,y)` within the viewport. The inclination of the string is specified in terms of differences of world coordinates making it easy to write text parallel to a line in a graph.

`x` (float, input):
x coordinate of reference point of string.

`y` (float, input):
y coordinate of reference point of string.

`dx` (float, input):

Together with `dy`, this specifies the inclination of the string. The baseline of the string is parallel to a line joining `(x,y)` to `(x+dx,y+dy)`.

`dy` (float, input):
Together with `dx`, this specifies the inclination of the string.

`just` (float, input):
Specifies the position of the string relative to its reference point. If `just=0`, the reference point is at the left and if `just=1`, it is at the right of the string. Other values of `just` give intermediate justifications.

`text` (char *, input):
The string to be written out.

```
plschr(def,scale);
```

This sets up the size of all subsequent characters drawn. The actual height of a character is the product of the default character size and a scaling factor.

`def` (float, input):
The default height of a character in millimetres, should be set to zero if the default height is to remain unchanged.

`scale` (float, input):
Scale factor to be applied to default to get actual character height.

```
plside3(x,y,z,ly,nx,ny,opt);
```

Draws sides on three dimensional plots. The sides extend from the plotted function down to the x-y plane. Should only be called after `plot3d`. The arguments passed to `plside3` should be the same as those used in `plot3d`.

`x` (float *, input):
Pointer to set of x coordinate values at which the function is evaluated.

`y` (float *, input):
Pointer to set of y coordinate values at which the function is evaluated.

`z` (float *, input):
Pointer to two dimensional array with set of function values.

`ly` (int, input):
Declared second dimension of `z` array.

`nx` (int, input):
Number of `x` values at which function is evaluated.

`ny` (int, input):
Number of `y` values at which function is evaluated.

`opt` (int, input):

Determines the way in which the sides are drawn:

- 1: Lines are drawn from `z` down to the y axis at each value of `y[j]`.
- 2: Lines are drawn from `z` down to the x axis at each value of `x[i]`.
- 3: Lines are drawn from `z` to both axes.

```
plsmaj(def,scale);
```

This sets up the length of the major ticks. The actual length is the product of the default length and a scaling factor as for character height.

`def` (float, input):
The default length of a major tick in millimetres, should be set to zero if the default length is to remain unchanged.

`scale` (float, input):
Scale factor to be applied to default to get actual tick length.

```
plsmin(def,scale);
```

This sets up the length of the minor ticks and the length of the terminals on error bars. The actual length is the product of the default length and a scaling factor as for character height.

`def` (float, input):
The default length of a minor tick in millimetres, should be set to zero if the default length is to remain unchanged.

`scale` (float, input):
Scale factor to be applied to default to get actual tick length.

```
plssym(def,scale);
```

This sets up the size of all subsequent symbols drawn by `plpoin` and `plsym`. The actual height of a symbol is the product of the default symbol size and a scaling factor as for the character height.

`def` (float, input):
The default height of a symbol in millimetres, should be set to zero if the default height is to remain unchanged.

`scale` (float, input):
Scale factor to be applied to default to get actual symbol height.

```
plstar(nx,ny);
```

Initializing the plotting package. The program prompts for the device number of the desired output device. The output device is divided into `nx` by `ny` sub-pages, each of which may be used independently. The subroutine `pladv` is used to advance from one subpage to the next.

`nx` (int, input):
Number of subpages to divide output page in the horizontal direction.

`ny` (int, input):
Number of subpages to divide output page in the vertical direction.

```
plstyl(nels,mark,space);
```

This sets up the line style for all lines subsequently drawn. A line consists of segments in which the pen is alternately down and up. The lengths of these segments are passed in the arrays `mark` and `space` respectively. The number of mark-space pairs is specified by `nels`. In order to return the line style to the default continuous line, `PLSTYL` should be called with `nels=0`.

`nels` (int, input):
The number of mark and space elements in a line. Thus a simple broken line can be obtained by setting `nels=1`. A continuous line is specified by setting `nels=0`.

`mark` (int *, input):
Pointer to array with the lengths of the segments during which the pen is down, measured in micrometres.

`space` (int *, input):
Pointer to array with the lengths of the segments during which the pen is up, measured in micrometres.

```
plsvpa(xmin,xmax,ymin,ymax);
```

Alternate routine to `plvpor` for setting up the viewport. This routine should be used only if the viewport is required to have a definite size in millimetres. The routine `plgspa` is useful for finding out the size of the current subpage.

`xmin` (float, input):
The distance of the left-hand edge of the viewport from the left-hand edge of the subpage in millimetres.

`xmax` (float, input):
The distance of the right-hand edge of the viewport from the left-hand edge of the subpage in millimetres.

`ymin` (float, input):
The distance of the bottom edge of the viewport from the bottom edge of the subpage in millimetres.

`ymax` (float, input):
The distance of the top edge of the viewport from the top edge of the subpage in millimetres.

```
plsym(n,x,y,code);
```

Marks out a set of `n` points at positions `(x[i],y[i])`, using the symbol defined by `code`. The code is interpreted as an index in the Hershey font tables.

`n` (int, input):
Number of points to be marked.

`x` (float *, input):
Pointer to array with set of x coordinate values for the points.

`y` (float *, input):
Pointer to array with set of y coordinate values for the points.

`code` (int, input):
Code number for the symbol to be plotted.

```
pltext();
```

Sets an interactive device to text mode, used in conjunction with `plgra` to allow graphics and text to be interspersed. This is not currently supported on the Amiga. All the labelling is drawn in vector mode.

```
plvpor(xmin,xmax,ymin,ymax);
```

Device-independent routine for setting up the viewport. This defines the viewport in terms of normalized subpage coordinates which run from 0.0 to 1.0 (left to right and bottom to top) along each edge of the current subpage. Use the alternate routine `plsvpa` in order to create a viewport of a definite size.

`xmin` (float, input):
The normalized subpage coordinate of the left-hand edge of the viewport.

`xmax` (float, input):
The normalized subpage coordinate of the right-hand edge of the viewport.

`ymin` (float, input):
The normalized subpage coordinate of the bottom edge of the viewport.

`ymax` (float, input):
The normalized subpage coordinate of the top edge of the viewport.

```
plvsta();
```

Sets up a standard viewport, leaving a left-hand margin of seven character heights, and four character heights around the other three sides.

```
plw3s(basex,basey,height,xmin,xmax,ymin,ymax,zmin,zmax,alt,az);
```

Sets up a window for a three-dimensional surface plot within the currently defined two-dimensional window. The enclosing box for the surface plot defined by `xmin`, `xmax`, `ymin`, `ymax`, `zmin` and `zmax` in user-coordinate space is mapped into a box of world coordinate size `basex` by `basey` by `height` so that `xmin` maps to `-basex/2`, `xmax` maps to `basex/2`, `ymin` maps to `-basey/2`, `ymax` maps to `basey/2`, `zmin` maps to `0` and `zmax` maps to `height`. The resulting world-coordinate box is then viewed by an observer at altitude `alt` and azimuth `az`. This routine must be called before `plbox3` or `plot3d`. See Section 3.10 on page 13 for a more complete description of three-dimensional plotting.

`basex` (float, input):
The x coordinate size of the world-coordinate box.

`basey` (float, input):
The y coordinate size of the world-coordinate box.

`height` (float, input):
The z coordinate size of the world-coordinate box.

`xmin` (float, input):
The minimum user x coordinate value.

`xmax` (float, input):
The maximum user x coordinate value.

`ymin` (float, input):
The minimum user y coordinate value.

`ymax` (float, input):
The maximum user y coordinate value.

`zmin` (float, input):
The minimum user z coordinate value.

`zmax` (float, input):
The maximum user z coordinate value.

`alt` (float, input):
The viewing altitude in degrees above the xy plane.

`az` (float, input):
The viewing azimuth in degrees. When `az=0`, the observer is looking face onto the zx plane, and as `az` is increased, the observer moves clockwise around the box when viewed from above the xy plane.

```
plwind(xmin,xmax,ymin,ymax);
```

Sets up the world coordinates of the edges of the viewport.

`xmin` (float, input):
The world x coordinate of the left-hand edge of the viewport.

`xmax` (float, input):
The world x coordinate of the right-hand edge of the viewport.

`ymin` (float, input):
The world y coordinate of the bottom edge of the viewport.

`ymax` (float, input):
The world y coordinate of the top edge of the viewport.