DOE/ET-53088-538 IFSR #538

The PLPLOT Plotting Library Programmer's Reference Manual Version 4.0

M.J. LEBRUN, G. FURNISH
Institute for Fusion Studies
The University of Texas at Austin
Austin, Texas 78712
and
T. RICHARDSON
Department of Electrical Engineering
Duke University
Durham, NC 27706

March 1992

Contents

1 Introduction **1**
1.1 The PLPLOT Plotting Library 1
1.2 Getting a copy of the PLPLOT package 2
1.3 Installing and Using the PLPLOT Library 3
1.4 Organization of this Manual 3
1.5 Credits 3

2 Simple Use of PLPLOT **6**
2.1 Plotting a Simple Graph 6
2.2 Initializing PLPLOT 6
2.3 Defining Plot Scales and Axes 7
2.4 Labelling the Graph 8
2.5 Drawing the Graph 9
2.5.1 Drawing Points 9
2.5.2 Drawing Lines or Curves 9
2.5.3 Writing Text on a Graph 10
2.5.4 Area Fills 10
2.5.5 More Complex Graphs (Histograms and Error Bars) 10
2.6 Finishing Up 11
2.7 In Case of Error 11

3 Advanced Use of PLPLOT **12**
3.1 Basic PLPLOT Concepts 12
3.2 Specifying the View Surface 12
3.3 Defining the Viewport 13
3.4 Defining the Window 15
3.5 Annotating the Viewport 15
3.6 The Routine `plenv` 16
3.7 Setting Line Attributes 16
3.8 Setting the Area Fill Pattern 17
3.9 Setting Character and Symbol Attributes 17
3.10 Escape Sequences in Text 18
3.11 Three Dimensional Surface Plots 19
3.12 Contour Plots 21
3.12.1 Contour Plots from C 21
3.12.2 Contour Plots from Fortran 23

4 PLPLOT Subroutine Reference . . . 25
pladv . . . 25
plancol . . . 25
plbin . . . 25
plbox . . . 26
plbox3 . . . 27
plclr . . . 29
plcol . . . 29
plcon0 . . . 30
plcon1 . . . 30
plcon2 . . . 31
plconf . . . 32
plcont . . . 33
plend . . . 34
plend . . . 34
plenv . . . 34
plerrx . . . 35
plerry . . . 35
plexit . . . 36
pl_exit . . . 36
plfill . . . 36
plfont . . . 37
plfontld . . . 37
plgfam . . . 37
plgfile . . . 37
plgfnam . . . 38
plgpage . . . 38
plgra . . . 38
plgspa . . . 38
plgstrm . . . 39
plgxax . . . 39
plgyax . . . 39
plgzax . . . 39
plhist . . . 40
pljoin . . . 40
pllab . . . 41
plline . . . 41
pllsty . . . 41
plmesh . . . 41
plmtex . . . 42
plot3d . . . 43
plpage . . . 44
plpat . . . 44
plpoin . . . 45
plprec . . . 45
plpsty . . . 45
plptex . . . 45
plsasp . . . 46

plschr 46
plsfam 47
plsfile 47
plsfnam 47
plsmaj 47
plsmin 48
plsori 48
plspage 48
plsstrm 49
plssym 49
plstar 49
plstart 50
plstyl 50
plsvpa 51
plsxax 51
plsyax 51
plsym 52
plszax 52
pltext 52
plvasp 52
plvpas 53
plvpor 53
plvsta 54
plw3d 54
plwid 55
plwind 55

A Programmer interface **56**
A.1 C language 56
A.2 Fortran language 57

B Output devices **60**
B.1 Driver functions 61
B.2 PLPLOT metafiles and plrender 62
B.3 Family file output 64
B.4 Interactive output devices 64

C System dependent usage and installation **66**
C.1 Unix-type systems 66
C.1.1 Linking 66
C.1.2 Installation 67
C.2 VMS 68
C.2.1 Linking 68
C.2.2 Installation 69
C.3 Amiga 69
C.3.1 Linking 69
C.3.2 Installation 70
C.3.3 IFF and printer driver 70

C.3.4 HP Plotter (or PLT: device) 71
C.3.5 Amiga window driver 71
C.4 OS/2 72
C.4.1 Compiler Support 72
C.4.2 Installation 72
C.4.3 Linking 73
C.4.4 Presentation Manager Support 73
C.5 MS-DOS 75
C.6 DOS 386 75

Chapter 1

Introduction

1.1 The PLPLOT Plotting Library

PLPLOT is a library of C functions that are useful for making scientific plots from a program written in C, C++, or Fortran. The PLPLOT library can be used to create standard x-y plots, semilog plots, log-log plots, contour plots, 3D plots, mesh plots, bar charts and pie charts. Multiple graphs (of the same or different sizes) may be placed on a single page with multiple lines in each graph. Different line styles, widths and colors are supported. A virtually infinite number of distinct area fill patterns may be used. There are almost 1000 characters in the extended character set. This includes four different fonts, the Greek alphabet and a host of mathematical, musical, and other symbols. The fonts can be scaled to any desired size. A variety of output devices are supported and new devices can be easily added by writing a small number of device dependent routines.

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.

The C version of PLPLOT was developed by Tony Richardson on a Commodore Amiga. In the process, several of the routines were rewritten to improve efficiency and some new features added. The program structure was changed somewhat to make it easier to incorporate new devices.

PLPLOT 4.0 is primarily the result of efforts by Maurice LeBrun and Geoff Furnish of University of Texas at Austin to extend and improve the previous work (PLPLOT 2.6b and 3.0, by Tony Richardson). PLPLOT 4.0 is currently used as the main graphics engine for TPC (Toroidal Particle Code), a large plasma simulation code developed at the IFS [1]. During this work we have found that PLPLOT compares well with "heavier" packages (read: expensive, slow) and is an excellent choice for scientists seeking an inexpensive (or free) but high quality graphics package that runs on many different computing platforms.

Some of the improvements in PLPLOT 4.0 include: the addition of several new routines to enhance usage from Fortran and design of a portable C to Fortran interface. Additional support was added for coordinate mappings in contour plots and some bugs fixed. New labelling options were added. The font handling code was made more flexible and portable. A portable PLPLOT metafile driver and renderer was developed, allowing one to create a generic graphics file and do the actual rendering later (even on a different system). The ability to create family output files was added. The internal code structure was dramatically reworked, with elimination of global variables (for a more robust package), the drivers rewritten to improve consistency, and the ability to maintain multiple output streams added. An XFig driver was added. Other contributions include Clair Nielsen's (LANL) X-window driver (very nice for high-speed color graphics) and tektronix file viewer. At present, Maurice LeBrun and Geoff Furnish are the active developers and maintainers of PLPLOT.

We have attempted to keep PLPLOT 4.0 backward compatible with previous versions of PLPLOT. However, some functions are now obsolete, and many new ones have been added (e.g. new contouring functions, variable get/set routines, functions that affect label appearance). Codes written in C that use PLPLOT must be recompiled including the new header file (`plplot.h`) before linking to the new PLPLOT library.

PLPLOT is currently known to work on the following systems: SUNOS, HP-UX, A/IX, DG/UX, UNICOS, Ultrix, VMS, Amiga/Exec, MS-DOS, OS/2, and NeXT, with more expected. The PLPLOT package is freely distributable, but *not* in the public domain. There have been various copyrights placed on the software; see section 1.5 for the full list and criteria for distribution.

We welcome suggestions on how to improve this code, especially in the form of user-contributed enhancements or bug fixes. If PLPLOT is used in any published papers, please include an acknowledgment or citation of our work, which will help us to continue improving PLPLOT. Please direct all communication to:

Dr. Maurice LeBrun Institute for Fusion Studies University of Texas Austin, TX 78712	Internet: mjl@fusion.ph.utexas.edu
Geoff Furnish Institute for Fusion Studies University of Texas Austin, TX 78712	Internet: furnish@fusion.ph.utexas.edu
Tony Richardson 184 Electrical Engineering Duke University Durham, NC 27706	Internet: amr@egr.duke.edu

The original version of this manual was written by Sze Tan.

1.2 Getting a copy of the PLPLOT package

At present, the only mechanism we are providing for distribution of the PLPLOT is by electronic transmission over the Internet. We encourage others to make it available to users without Internet access. PLPLOT may be obtained by `ftp` from `hagar.ph.utexas.edu` (128.83.179.27). Login as user `anonymous`, set file transfer type to binary, and get the newest plplot archive in the `pub/` subdirectory. We will provide PLPLOT in both `zoo` and `tar` archives; get whichever one you prefer.

1.3 Installing and Using the PLPLOT Library

The installation procedure is by necessity system specific; installation notes for each system are provided in Appendix C. The procedure requires that all of the routines be compiled and they are then usually placed in a linkable library.

After the library has been created, you can write your main program to make the desired PLPLOT calls. Example programs in both C and Fortran are included as a guide (if calling from C, you must include `plplot.h` into your program; see Appendix A for more details). Plots generated from the example programs are shown at the end of this work.

You will then need to compile your program and link it with the PLPLOT library(s). Again, please refer to the documentation specific to your system for this. Note that there may be more than one library available to link with, such as single or double precision, with or without X window libraries, IEEE floating point or Motorola FFP, etc. Make sure you link to the correct one for your program.

1.4 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. Chapter 4 is the reference section of the manual, containing an alphabetical list of the user-accessible PLPLOT functions with detailed descriptions.

Because the PLPLOT kernel is written in C, standard C syntax is used in the description of each PLPLOT function. The C and Fortran language interfaces are discussed in Appendix A; look there if you have difficulty interpreting the call syntax as described in this manual. The meaning of function (subroutine) arguments is typically the same regardless of whether you are calling from C or Fortran (but there are some exceptions to this). The arguments for each function are usually specified in terms of PLFLT and PLINT — these are the internal PLPLOT representations for integer and floating point, and are typically a long and a float (or an INTEGER and a REAL, for Fortran programmers). See Appendix A for more detail.

Also, you can use PLPLOT from C++ just as you would from C. No special classes are available at this time, just use it as any other procedural type library. Simply include `plplot.h`, and invoke as you would from C.

The output devices supported by PLPLOT are listed in Appendix B, along with description of the device driver–PLPLOT interface, metafile output, family files, and vt100/tek4010 emulators. In Appendix C the usage and installation for each system supported by PLPLOT is described (not guaranteed to be entirely up-to-date; check the release notes to be sure).

1.5 Credits

PLPLOT 4.0 was created through the effort of many individuals and funding agencies. We would like to acknowledge the support (financial and otherwise) of the following institutions:

The Institute for Fusion Studies, University of Texas at Austin

The Scientific and Technology Agency of Japan

Japan Atomic Energy Research Institute

Duke University

Universite de Nice

National Energy Research Supercomputer Center

Los Alamos National Labs

PLPLOT was first derived from the excellent PGPLOT graphics package by T. J. Pearson. All parts of PLPLOT not explicitly marked by a copyright are assumed to derive sufficiently from the original to be covered by the PGPLOT copyright:

The startup code for plrender.c is from `xterm.c` of the X-windows Release 5.0 distribution, and we reproduce its copyright here:

Thanks are also due to those people contributing various parts to PLPLOT, including:

Sam Paolucci (postscript driver)

Clair Nielsen (X driver and tektronix file viewer)

Tom Rokicki (IFF driver and Amiga printer driver)

Finally, thanks to all those who submitted bug reports and other suggestions.

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. PLPLOT only needs to be initialized once (via `plstar` or a different startup routine), unless it is desired to switch output devices between pages.

2.2 Initializing 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 (PLINT, input):
The number of plots to a page. The page is divided into **nx** $\times$ **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, read page 13 in Section 3.3 which discusses viewports.

An alternate startup routine **plstart** behaves identically to **plstar** except that the device name is input by a keyword argument (keywords are listed for each device in Appendix B). This style of call is convenient if the user always outputs to the same device. If the requested device is not available, or if the input string is empty or begins with "?", the prompted startup of **plstar** is used. The syntax for **plstart** is:

```
plstart(devname, nx, ny);
```

devname (char *, input):
The device name, in the form of a keyword for that device.

nx, ny (PLINT, input):
The number of plots to a page, as in **plstar**.

The majority of calls to PLPLOT are made after initializing it by one of the routines listed above, however, a few must be made *before* the initialization in order to correctly set up the subsequent plots (the description in the reference section identifies these setup calls).

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 (PLFLT, input):
The left and right limits for the horizontal axis.

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

just (PLINT, 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` (PLINT, 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 53), **plvasp** (page 52), **plvpas** (page 53), **plwind** (page 55), **plbox** (page 26), and routines for manipulating axis labelling **plgxax** (page 39) through **plszax** (page 52), 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 centered 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 **plmtex** (page 42). See Section 2.5.3 for information about the function **plptex** which writes labels within a graph, and section 3.5 which discusses floating point formats.

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` (PLINT, input):
> the number of points to plot.
>
> `x, y` (PLFLT *, input):
> pointers to arrays of the coordinates of the `n` points.
>
> `code` (PLINT, 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, style and width. See page 16 in Section 3.7, page 17 in Section 3.7 and page 16 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` (PLINT, input):
> the number of points.
>
> `x, y` (PLFLT *, input):
> pointers to arrays with coordinates of the `n` points.

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

> `x1, y1` (PLFLT, input):
> coordinates of the first point.
>
> `x2, y2` (PLFLT, 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` (PLFLT, input):
: coordinates of the reference point.

`dx, dy` (PLFLT, 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` (PLFLT, 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 Area Fills

Area fills are done in the currently selected color, line style, line width and pattern style.
`plfill` fills a polygon. The polygon consists of `n` vertices which define the polygon.

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

`n` (PLINT, input):
: the number of vertices.

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

2.5.5 More Complex Graphs (Histograms and Error Bars)

Functions `plbin` (page 25) and `plhist` (page 40) are provided for drawing histograms, and functions `plerrx` (page 35) and `plerry` (page 35) 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 34) to close any output plot files and to free up resources. For devices that have separate graphics and text modes, **plend** resets the device to text mode.

2.7 In Case of Error

If a fatal error is encountered during execution of a PLPLOT routine then **plexit** (page 36) is called. This routine prints an error message, does resource recovery, calls **pl_exit** (page 36) and then exits. The default **pl_exit** routine does nothing, but the user may wish to supply his/her own version of **pl_exit** (for C programs only).

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 can be a terminal screen or a sheet of paper in the output device (or a portion thereof, if subpages are being used). 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

Before any actual plotting calls are made, a graphics program initialize PLPLOT, i.e. specify the output device and how this device is to be divided up for graph plotting. There are at present two routines recommended for this purpose, `plstar` (page 49), which prompts at the console for the output device type, and `plstart` (page 50) which expects to be supplied the keyword of the device as an argument. The old routine `plbeg` can also be used for initialization, but use of this routine is discouraged since it requires the device *number* which depends on the particular device drivers installed for your system (`plbeg` has only been retained for compatibility with old codes, and will not be discussed further).

Besides selecting the device, `plstar` and `plstart` 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 (or a new piece of paper 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.

A new addition to PLPLOT is the ability to write to multiple output streams. An output stream corresponds to a single logical device to which one plots independent of all other streams. The function **plsstrm** (page 49) is used to switch between streams (note one may only write to one output stream at a time). At present, an output stream is not limited by the type of device (however it may not be wise to attempt opening two terminal devices). Eventually it will be possible to open multiple output devices per stream, with plots going to each device simultaneously, however this feature was not ready in time for the PLPLOT 4.0 release. An example usage for the creation of multiple streams is as follows:

```
#include "plplot.h"

main()
{
    int nx = 2, ny = 2;

    plstart("xterm", nx, ny);

    <plots for stream 0>

    plsstrm(1);
    plsfnam("tst.plm");
    plstart("plmeta", nx, ny);

    <plots for stream 1>

    plsstrm(0);

    <plots for stream 0>
```

and so on, for sending output simultaneously to an xterm and a metafile. The default stream corresponds to stream number zero. At present, the majority of output drivers can only be used by a single stream (exceptions include the metafile driver and X-window driver). Also see example program 14 (note: only the C version is available, although it can be done equally well from Fortran).

At the end of a plotting program, it is important to close the plotting device by calling **plend** (page 34). This flushes any internal buffers and frees any memory that may have been allocated, for all open output streams. You may call **plend1** (page 34) to close the plotting device for the current output stream only. Note that if PLPLOT is initialized more than once during a program to change the output device, an automatic call to **plend1** is made before the new device is opened for the given stream.

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 (the viewport). 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 **plstart** or **plstar**), and use one of the viewport specification subroutines below to place the plot in the desired position on the page.

There are four methods for specifying the viewport size, using the subroutines **plvpor** (page 53), **plsvpa** (page 51), **plvasp** (page 52), and **plvpas** (page 53). Each of these has the format:

```
plvpor(xmin, xmax, ymin, ymax);
plsvpa(xmin, xmax, ymin, ymax);
plvasp(aspect);
plvpas(xmin, xmax, ymin, ymax, aspect);
```

where in the case of **plvpor** and **plvpas**, 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.

The routine **plvasp** gives the largest viewport with the given aspect ratio that fits in the current subpage (i.e. the ratio of the length of the y axis to that of the x axis is equal to **aspect**). It also allocates space on the left and top of the viewport for labels.

The routine **plvpas** gives the largest viewport with the given aspect ratio that fits in the specified region (specified with normalized subpage coordinates, as with **plvpor**). This routine is functionally equivalent to **plvpor** when a "natural" aspect ratio is chosen (done by setting **aspect** to 0.0). Unlike **plvasp**, this routine reserves no extra space at the edges for labels.

To help the user call **plsvpa** correctly, the routine **plgspa** (page 38) 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 54) 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).

Another way to get a specified aspect ratio is via the routine **plsasp** (page 46), which sets the global aspect ratio and must be called prior to **plstar**. An aspect ratio of 0.0 corresponds to "natural" dimensions (i.e. fill the page); any positive value will give the specified aspect ratio. This scaling of plots is actually done in the driver, and so may not work for all output devices (note that **plrender** is capable of scaled aspect ratio plots to any device whether that device supports scaling or not). In such scaled plots, absolute plotting is done in the scaled coordinate system.

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 55) 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 $0 \leq I_C \leq 10.0$ mA and $0 \leq V_{CE} \leq 12.0$ 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 Annotating the Viewport

The routine **plbox** (page 26) 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 41) 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 18) which allow subscripts and superscripts to be used in text. They are described more fully later in this chapter.

The appearance of axis labels may be further altered by auxiliary calls to **plprec** (page 45), **plschr** (page 46), **plsxax** (page 51), **plsyax** (page 51), and **plszax** (page 52). The routine **plprec** is used to set the number of decimal places precision for axis labels, while **plschr** modifies the heights of characters used for the axis and graph labels. Routines **plsxax**, **plsyax**, and **plszax** are used to modify the **digmax** setting for each axis, which affects how floating point labels are formatted.

The **digmax** variable represents the maximum field width for the numeric labels on an axis (ignored if less than one). If the numeric labels as generated by PLPLOT exceed this width, then PLPLOT automatically switches to floating point representation. In this case the exponent will be placed at the top left for a vertical axis on the left, top right for a vertical axis on the right, and bottom right for a horizontal axis.

For example, let's suppose that we have set **digmax** = 5 via **plsyax**, and for our plot a label is generated at y = 0.0000478. In this case the actual field width is longer than **digmax**, so PLPLOT switches to floating point. In this representation, the label is printed as simply 4.78 with the 10^{-5} exponent placed separately.

The determination of maximum length (i.e. **digmax**) for fixed point quantities is complicated by the fact that long fixed point representations look much worse than the same sized floating point representation. Further, a fixed point number with magnitude much less than one will actually gain in precision when written as floating point. There is some compensation for this effect built into PLPLOT, thus the internal representation for number of digits kept (**digfix**) may not always match the user's specfication (via **digmax**). However, it will always be true that **digfix** $\leq$ **digmax**. The PLPLOT defaults are set up such that good results are usually obtained without user intervention.

Finally, after the call to **plbox**, the user may call routines **plgxax** (page 39), **plgyax** (page 39), or **plgzax** (page 39) to obtain information about the window just drawn. This can be helpful when deciding where to put captions. For example, a typical usage would be to call **plgyax** to get the value of **digits**, then offset the y axis caption by that amount (plus a bit more) so that the caption "floats" just to the outside of the numeric labels. Note that the **digits** value for each axis for the current plot is not correct until *after* the call to **plbox** is complete.

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, allowing the user to control line color, width 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 color is set using the routine **plcol** (page 29). The argument is ignored for devices which can only plot in one color, although some terminals support line erasure by plotting in color zero.

Line width is set using **plwid** (page 55). This option is not supported by all devices.

Line style is set using the routine plstyl (page 50) or pllsty (page 41). 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 are the number of elements in the line, followed by two pointers to integer arrays specifying the mark and space lengths in micrometers. 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. You can use pllsty to choose between 8 different predefined styles.

3.8 Setting the Area Fill Pattern

The routine plpat (page 44) can be used to set the area fill pattern. The pattern consists of 1 or 2 sets of parallel lines with specified inclinations and spacings. The arguments to this routine are the number of sets to use (1 or 2) followed by two pointers to integer arrays (of 1 or 2 elements) specifying the inclinations in tenths of a degree and the spacing in micrometers. (The inclination should be between -900 and 900.) Thus to specify an area fill pattern consisting of horizontal lines spaced 2 mm apart use:

```
*inc = 0;
*del = 2000;
plpat(1, inc, del);
```

To set up a symmetrical crosshatch pattern with lines directed 30 degrees above and below the horizontal and spaced 1.5 mm apart use:

```
*inc = 300;
*(inc+1) = -300;
*del = 1500;
*(del+1) = 1500;
plpat(2, inc, del);
```

The routine plpsty (page 45) can be used to select from 1 of 8 predefined patterns.

The area fill routines also use the current line style, width and colors to give a virtually infinite number of different patterns!

3.9 Setting Character and Symbol Attributes

There are two character sets included with PLPLOT. These are known as the standard and extended character sets respectively. The standard character set is a subset of the extended set. It contains 177 characters including the ascii characters in a normal style font, the greek alphabet and several plotter symbols. The extended character set contains almost 1000 characters, including four font styles, and several math, musical and plotter symbols.

The standard character set is loaded into memory automatically when `plstar` or `plstart` is called. The extended character set is loaded by calling `plfontld` (page 37). The extended character set requires about 50 kbytes of memory, versus about 5 kbytes for the standard set. `plfontld` can be used to switch between the extended and standard sets (one set is unloaded before the next is loaded). `plfontld` can be called before `plstar`.

When the extended character set is loaded there are four different font styles to choose from. In this case, the routine `plfont` (page 37) sets up the default font for all character strings. It may be overridden for any portion of a string by using an escape sequence within the text, as described below. This routine has no effect when the standard font set is loaded. The default font (1) is simple and fastest to draw; 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 46) is used to set up the size of subsequent 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 `plstart`, 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 use 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 49) 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 47) and `plsmin` (page 48).

3.10 Escape Sequences in Text

The routines which draw text all allow you to include escape sequences in the text to be plotted. These are character sequences that are interpreted as instructions to change fonts, draw superscripts and subscripts, draw non-ASCII (e.g. Greek), and so on. All escape sequences start with a number symbol (`#`).

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)
- `##`: number symbol
- `#+`: toggle overline mode

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

- **#-**: 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

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

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.

3.11 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 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 centered 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 55) or `plenv` (page 34) is used in the usual way to establish the size of the two-dimensional window. The routine `plw3d` (page 54) 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 43).

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

```
xmin2d = -2.5;
xmax2d =  2.5;
ymin2d = -2.5;
ymax2d =  4.0;
plenv(xmin2d, xmax2d, ymin2d, ymax2d, 0, -2);
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;
side = 1;
plw3d(basex, basey, height, xmin, xmax, ymin, ymax, zmin, zmax, alt, az);
plot3d(x, y, z, nx, ny, opt, side);
```

The values of the function are stored in a two-dimensional array `z[][]` where the array element `z[i][j]` contains the value of the function at the point x_i, y_j. (The two-dimensional array z is a vectored array instead of a fixed size array. z points to an array of pointers which each point to a rwo of the matrix.) 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. If side is equal to one then sides are drawn on the figure so that the graph doesn't appear to float.

The routine plmesh (page 41) is similar to plot3d, except that it is used for drawing mesh plots. Mesh plots allow you to see both the top and bottom sides of a surface mesh, while 3D plots allow you to see the top side only (like looking at a solid object). The side option is not available with plmesh.

Labelling a three-dimensional or mesh 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 27) thus combines the functions of box drawing and labelling. Its parameters are described more fully in Chapter 4.

3.12 Contour Plots

Several routines are available in PLPLOT which perform 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. Further, one may specify arbitrary coordinate mappings from array indices to world coordinates, such as for contours in a polar coordinate system. In this case it is best to draw the distinction between a C and Fortran language caller, so these are handled in turn.

3.12.1 Contour Plots from C

There are two routines callable from C for plotting contours, plcont (page 33) and plconf (page 32). Routine plcont has the form

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

where z is the two-dimensional array of size nx $\times$ ny containing samples of the function to be contoured. (z is a vectored two-dimensional array as described in the previous section. It is *not* a fixed-size two-dimensional array.) 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. Routine plconf has the form

```
plconf(z, nx, ny, kx, lx, ky, ly, clevel, nlevel, pltr, xg, yg)
```

with all arguments identical to those in plcont except for the pointers to transformation arrays xg and yg. These are used as input to the transformation function pltr.

The path of each contour is initially computed in terms of the values of the array indicies which range from 0 to nx-1 in the first index and from 0 to ny-1 in the second index. Before these can be drawn in the current window (see page 15 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` (the case for `plconf` is slightly different and discussed below). This function pointer is the last argument `pltr`. This function must be declared as type `void` in the module which calls `plcont`. This transformation function must have the parameter list

```
void pltr(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 indices and world coordinates can be expressed as a linear transformation. A routine is provided within the library which can be passed to `plcont` as `pltr`. This transformation routine is

```
#include "plplot.h"

#ifdef PLSTDC
void
xform(PLFLT x, PLFLT y, PLFLT * tx, PLFLT * ty)
#else
void
xform(x, y, tx, ty)
PLFLT x, y, *tx, *ty;
#endif
{
    *tx = tr[0] * x + tr[1] * y + tr[2];
    *ty = tr[3] * x + tr[4] * y + tr[5];
}
```

and should be compiled as part of the user's code (it is *not* normally linked in with the PLPLOT library since the `tr` data array is unspecified).

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. If no transformation is required, you can simply set `tr[0]=tr[4]=1.`, with other entries set to 0.. Alternately you can pass the built-in function `pltr0` to `plcont`; this function merely sets `*tx` to `x` and `*ty` to y, resulting in no net transformation.

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 (0, 0) in array `z` to map to (0.0, -8.0) and the index (20, 40) 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

$$tx = 0.2x$$
$$ty = 0.4y - 8.$$

and so `tr[0] = 0.2`, `tr[1] = 0.0`, `tr[2] = 0.0`, `tr[3] = 0.0`, `tr[4] = 0.4`, `tr[5] = -8.`.

For routine **plconf**, the user must pass pointers to transformation arrays as well as the function pointer. The use of transformation arrays is generally the most convenient and flexible way to map coordinates, since the mapping function (which only handles interpolation) need

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`, performing a page advance if there are no remaining subpages on the current page. If subwindowing isn't being used, `pladv(0)` will always advance the page. If `sub>0`, PLPLOT switches to the specified subpage. Note that this allows you to overwrite a plot on the specified subpage; if this is not what you intended, use `plclr` followed by `plpage` to first advance the page. This routine is called automatically (with `sub=0`) by `plenv`, but if `plenv` is not used, `pladv` must be called after initializing PLPLOT but before defining the viewport.

`sub` (PLINT, 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.

`plancol(color, name)`

Allocates the specified named color and assigns it to the color numbered `color`. Supported by the X-window driver only at present. See `plcol` for more information.

`color` (PLINT, input):
Specifies the color number as set by the `plcol` command.

`name` (char *, input):
Specifies the color name. The RGB values corresponding to the given color are those given by the X11 RGB data base (`rgb.txt`).

`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 `center` is false, `x[i]` is the left-hand edge of the `i'th` bin, and if `center` 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` (PLINT, input):
Number of bins (i.e., number of values in x and y arrays.)

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

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

`cen` (PLINT, 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 center 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 `x=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` (PLFLT, 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` (PLINT, 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` (PLINT, 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 3.11 on page 19 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` (PLFLT, 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` (PLINT, 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` (PLFLT, 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` (PLINT, 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.
- `d` : Draws grid lines parallel to the x-y plane behind the figure. These lines are not drawn until after **plot3d** or **plmesh** are called because of the need for hidden line removal.
- `i` : Inverts tick marks, so they are drawn away from the center.
- `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` (PLFLT, 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` (PLINT, 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. See `plpage` for more information.

`plcol(color)`

Sets the color for subsequent lines. You may specify which colors are associated with which number by the `plancol` routine if supported by the output driver.

`color` (PLINT, input):
: Integer representing the color. The defaults at present are (these may change):

0	coral
1	red
2	yellow
3	green
4	aquamarine
5	pink
6	wheat
7	grey
8	brown
9	blue
10	BlueViolet
11	cyan
12	turquoise
13	magenta
14	salmon
15	white (default foreground)
16	black (default background)

`plcon0(z, nx, ny, kx, lx, ky, ly, clevel, nlevel)` (Fortran)

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. See Section 3.12 for more information.

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

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

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

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

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

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

NOTE: this function is intended for use from a Fortran caller only. The C user should instead call `plcont` using the built-in transformation function `pltr0` for the same capability.

`plcon1(z, nx, ny, kx, lx, ky, ly, clevel, nlevel, xg, yg)` (Fortran)

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. The arrays **`xg`** and **`yg`** are used to specify the transformation between array indices and world coordinates. See Section 3.12 for more information.

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

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

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

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

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

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

`xg`, `yg` (PLFLT *, input):
Pointers to arrays which specify the transformation from array indices to world coordinates. These must be one-dimensional arrays, used for a transformation of the form: `tx=f(x), ty=f(y)`. Function values at locations between grid points are obtained via linear interpolation.

NOTE: this function is intended for use from a Fortran caller only. The C user should instead call `plconf` using the built-in transformation function `pltr1` for the same capability.

`plcon2(z, nx, ny, kx, lx, ky, ly, clevel, nlevel, xg, yg)` (Fortran)

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. The arrays **`xg`** and **`yg`** are used to specify the transformation between array indices and world coordinates. See Section 3.12 for more information.

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

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

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

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

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

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

`xg`, `yg` (PLFLT *, input):
: Pointers to arrays which specify the transformation from array indices to world coordinates. These must be two-dimensional arrays, used for a transformation of the form: `tx=f(x, y)`, `ty=f(x, y)`. Function values at locations between grid points are obtained via linear interpolation.

NOTE: this function is intended for use from a Fortran caller only. The C user should instead call `plconf` using the built-in transformation function `pltr2` for the same capability.

`plconf(z, nx, ny, kx, lx, ky, ly, clevel, nlevel, pltr, xg, yg)` (C)

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 `pltr` is used to map indicies within the array to the world coordinates, using the transformation arrays `xg` and `yg`. See Section 3.12 for more information.

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

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

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

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

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

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

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

`pltr(x, y, tx, ty, xg, yg, nx, ny)`

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

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

xg, yg (PLFLT *, input):
    Pointers to transformation arrays.

nx, ny (PLINT, input):
    Array dimensions in x and y, respectively.

xg, yg (PLFLT *, input):
    Pointers to arrays which define the transformation from (x, y) to (tx, ty), used as input to the transformation function pltr.

Two transformation functions are provided: pltr1, for one dimensional mappings (i.e. tx=f(x), ty=f(y)), and pltr2, for two dimensional mappings (i.e. tx=f(x, y), ty=f(x, y)). Function values at locations between grid points are obtained via linear interpolation.

NOTE: this routine cannot be called directly from Fortran. Use plcon0, plcon1, or plcon2 instead.

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

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

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 **pltr** is used to map indicies within the array to the world coordinates. See Section 3.12 for more information.

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

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

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

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

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

nlevel (PLINT, input):
    Number of contour levels to draw.

`pltr` (void *, input):

Pointer to function that defines transformation between indicies in array **z** and the world coordinates (C only). Two transformation functions are provided: **xform**, for simple linear mappings, and **pltr0**, if no mapping is required. **xform** is not actually compiled into the PLPLOT library due to the need for external data; instead you must include it as part of your program. The function should have the form

`pltr(x, y, tx, ty)`

> `x, y` (PLFLT, input):
>
> Specifies the position in the array through which the contour runs in terms of the array indicies.
>
> `tx, ty` (PLFLT *, output):
>
> Pointers to the world coordinates corresponding to the point `(x, y)`.

`tr` (PLFLT *, input):

Pointer to 6-element array specifying coefficients to use in the **xform** transformation function (Fortran only). Since this is somewhat inflexible, the user is recommended to call either of **plcon0**, **plcon1**, or **plcon2** instead.

`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.

`plend1()`

Ends a plotting session for the current output stream only. See **plsstrm** for more info.

`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**, **plvpor**, or **plvasp** for setting up the viewport, **plwind** for defining the window, and **plbox** for drawing the box.

xmin (PLFLT, input):

Value of x at left-hand edge of window.

xmax (PLFLT, input):

Value of x at right-hand edge of window.

ymin (PLFLT, input):

Value of y at bottom edge of window.

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

just (PLINT, 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 (PLINT, 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 (PLINT, input):
Number of error bars to draw.

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

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

y (PLFLT *, 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 (PLINT, input):
Number of error bars to draw.

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

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

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

`plexit(message)`

This routine is called in case an error is encountered during execution of a PLPLOT routine. It prints the error message, tries to release allocated resources, calls **pl_exit** and then exits. If cleanup needs to be done in the driver program then the user may want to supply his/her own version of **pl_exit**.

message (char *, input):
Error message.

`pl_exit()`

This routine does nothing. It is intended to be replaced by a user routine of the same name so that any user allocated resources can be released in the case of an error (C programs only). See `plexit` above.

`plfill(n, x, y)`

Fills the polygon defined by the `n` points `(x[i], y[i])` using the pattern defined by `plpsty` or `plpat`. The routine will automatically close the polygon between the last and first vertices. If multiple closed polygons are passed in `x` and `y` then `plfill` will fill in between them.

n (PLINT, input):
Number of vertices in polygon.

x (PLFLT *, input):
Pointer to array with x coordinates of vertices.

y (PLFLT *, input):
Pointer to array with y coordinates of vertices.

`plfont(font)`

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

This routine has no effect unless the extended character set is loaded (see **plfontld**).

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

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

`plfontld(set)`

Sets the character set to use for subsequent character drawing. May be called before calling initializing PLPLOT.

`set` (PLINT, input):
Specifies the character set to load:

- 0: Standard character set
- 1: Extended character set

`plgfam(fam, num, bmax)`

Gets information about current family file, if familying is enabled. See Section B.3 for more information.

`fam` (PLINT *, output):
Pointer to variable with the boolean family flag value. If nonzero, familying is enabled.

`num` (PLINT *, output):
Pointer to variable with the current family file number.

`bmax` (PLINT *, output):
Pointer to variable with the maximum file size (in bytes) for a family file.

`plgfile(file)`

Gets the current output file handle, if applicable.

`file` (FILE *, output):
File pointer to current output file.

NOTE: cannot be called from Fortran.

`plgfnam(fnam)`

Gets the current output file name, if applicable.

`fnam` (char *, output):
Pointer to file name string.

`plgpage(xp, yp, xleng, yleng, xoff, yoff)`

Gets the current page configuration.

`xp` (PLINT *, output):
Pointer to number of pixels in x.

`yp` (PLINT *; output):
Pointer to number of pixels in y.

`xleng` (PLINT *, output):
Pointer to x page length value.

`yleng` (PLINT *, output):
Pointer to y page length value.

`xoff` (PLINT *, output):
Pointer to x page offset.

`yoff` (PLINT *, output):
Pointer to y page offset.

`plgra()`

Sets an interactive device to graphics mode, used in conjunction with `pltext` to allow graphics and text to be interspersed. On a device which supports separate text and graphics windows, this command causes control to be switched to the graphics window. If already in graphics mode, this command is ignored. It is also ignored on devices which only support a single window or use a different method for shifting focus. See also `pltext`.

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

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

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

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

`plgstrm(strm)`

Gets the number of the current output stream. See also **plsstrm**.

strm (PLINT *, output):
Pointer to current stream value.

`plgxax(digmax, digits)`

Returns current values of the **digmax** and **digits** flags for the x axis. **digits** is updated after the plot is drawn, so so this routine should only be called *after* the call to plbox (or plbox3) is complete. See section 3.5 for more information.

digmax (PLINT *, output):
Pointer to variable with the maximum number of digits. If nonzero, the printed label will be switched to a floating point representation when the number of digits exceeds digmax.

digits (PLINT *, output):
Pointer to variable with the actual number of digits for the numeric labels (x axis) from the last plot.

`plgyax(digmax, digits)`

Identical to **plgxax**, except that arguments are flags for y axis. See the description of **plgxax** for more detail.

`plgzax(digmax, digits)`

Identical to **plgxax**, except that arguments are flags for z axis. See the description of **plgxax** for more detail.

`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 (PLINT, input):
: Number of data points.

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

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

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

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

oldwin (PLINT, 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 (PLFLT, input):
: x coordinate of first point.

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

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

y2 (PLFLT, 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 (PLINT, input):
Number of points to join.

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

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

`pllsty(n)`

This sets the line style according to one of eight predefined patterns (also see **plstyl**).

n (PLINT, input):
Integer value between 1 and 8.

`plmesh(x, y, z, nx, ny, opt)` (C)

`plmesh(x, y, z, nx, ny, opt, mx)` (Fortran)

Plots a surface mesh within the environment set up by **plw3d**. The surface is defined by the two-dimensional array `z[][]`, 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 3.11 on page 19 for further details.

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

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

z (PLFLT **, input):
Pointer to a vectored two-dimensional array with set of function values.

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

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

opt (PLINT, 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.

mx (PLINT, input):
Length of array in x direction, for plotting subarrays (Fortran only).

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

just (PLFLT, 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, nx, ny, opt, side)` (C)
`plot3d(x, y, z, nx, ny, opt, side, mx)` (Fortran)

Plots a three dimensional surface plot within the environment set up by **plw3d**. The surface is defined by the two-dimensional array `z[][]`, 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 3.11 on page 19 for further details.

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

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

z (PLFLT **, input):
Pointer to a vectored two-dimensional array with set of function values.

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

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

opt (PLINT, 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.

side (PLINT, input):
Flag to indicate whether or not "sides" should be draw on the figure. If **side=0** no sides are drawn, otherwise the sides are drawn.

mx (PLINT, input):
Length of array in x direction, for plotting subarrays (Fortran only).

plpage()

Begins a new page. For a file driver, the output file is opened if necessary. Advancing the page via **plclr** and **plpage** is useful when a page break is desired at a particular point when plotting to subpages. Another use for **plclr** and **plpage** is when plotting pages to different files, since you can manually set the file name (or file handle) by calling **plsfnam** or **plsfile** after the call to **plclr** (in fact some drivers may only support a single page per file, making this a necessity). One way to handle this case automatically is to page advance via **pladv**, but enable familying (see **plsfam**) with a small byte per file limit so that a new family member file will be created on each page break.

Except for one of these circumstances, it is best to advance the page or subpage simply by the call **pladv(0)**.

plpat(nlin, inc, del)

Sets the area fill pattern. The pattern consists of 1 or 2 sets of parallel lines with specified inclinations and spacings. The arguments to this routine are the number of sets to use (1 or 2) followed by two pointers to integer arrays (of 1 or 2 elements) specifying the inclinations in tenths of a degree and the spacing in micrometers. (also see **plpsty**)

nlin (PLINT, input):
Number of sets of lines making up the pattern, either 1 or 2.

inc (PLINT *, input):
Pointer to array with **nlin** elements. Specifies the line inclination in tenths of a degree. (Should be between -900 and 900).

del (PLINT *, input):
Pointer to array with **nlin** elements. Specify the spacing in micrometers between the lines making up the pattern.

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

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

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

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

`plprec(set, prec)`

Sets the number of places after the decimal point in numeric labels.

`set` (PLINT, input):
If **`set`** is equal to 0 then PLPLOT automatically determines the number of places to use after the decimal point in numeric labels (like those used to label axes). If **`set`** is 1 then **`prec`** sets the number of places.

`prec` (PLINT, input):
The number of characters to draw after the decimal point in numeric labels.

`plpsty(n)`

Select one of eight predefined area fill patterns to use (also see **`plpat`**).

`n` (PLINT *, input):
The desired pattern.

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

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

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

`just` (PLFLT, 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.

`plsasp(aspect)`

This sets up the aspect ratio for all subsequent pages, if supported by the driver. This routine, if used, must be called before initializing PLPLOT. This routine at present does not always give good results; the user is encouraged to instead plot to viewports of definite aspect ratios via the **plvasp** or **plvpas** routines. Note that **plrender** is capable of scaled aspect ratio plots to any device whether that device supports scaling or not.

`aspect` (PLFLT, input):
: Ratio of the page length in y to the page length in x.

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

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

`plsfam(fam, num, bmax)`

Sets variables dealing with output file familying. Does nothing if familying not supported by the driver. This routine, if used, must be called before initializing PLPLOT. See Section B.3 for more information.

`fam` (PLINT, input):
Family flag (boolean). If nonzero, familying is enabled.

`num` (PLINT, input):
Current family file number.

`bmax` (PLINT, input):
Maximum file size (in bytes) for a family file.

`plsfile(file)`

Sets the current output file handle, if applicable. If the file has has not been previously opened and is required by the driver, the user will be prompted for the file name. This routine, if used, must be called before initializing PLPLOT.

`file` (FILE *, output):
File pointer.

NOTE: cannot be called from Fortran.

`plsfnam(fnam)`

Sets the current output file name, if applicable. If the file name has not been specified and is required by the driver, the user will be prompted for it. If using the X-windows output driver, this sets the display name. This routine, if used, must be called before initializing PLPLOT.

`fnam` (char *, output):
Pointer to file name string.

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

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

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

`plsori(ori)`

Sets the current orientation. If `ori` is equal to zero (default) then landscape is used (x axis is parallel to the longest edge of the page), otherwise portrait is used. This option is not supported by all output drivers (in particular, most interactive screen drivers ignore the orientation). This routine, if used, must be called before initializing PLPLOT.

`ori` (PLINT, output):
Orientation value.

`plspage(xp, yp, xleng, yleng, xoff, yoff)`

Sets the page configuration (optional). Not all parameters recognized by all drivers. The X-window driver uses the length and offset parameters to determine the window size and location. This routine, if used, must be called before initializing PLPLOT.

`xp` (PLINT *, output):
Number of pixels, x.

`yp` (PLINT *; output):
Number of pixels, y.

`xleng` (PLINT *, output):
Page length, x.

`yleng` (PLINT *, output):
Page length, y.

`xoff` (PLINT *, output):
Page offset, x.

`yoff` (PLINT *, output):
Page offset, y.

`plsstrm(strm)`

Sets the number of the current output stream. The stream number defaults to 0 unless changed by this routine. The first use of this routine must be followed by a call initializing PLPLOT (e.g. **plstar**).

`strm` (PLINT *, output):
Pointer to current stream number.

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

`scale` (PLFLT, 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 keyword or number of the desired output device. Hitting a RETURN in response to the prompt is the same as selecting the first device. If only one device is enabled when PLPLOT is installed, **plstar** will issue no prompt. 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 (PLINT, input):
Number of subpages to divide output page in the horizontal direction.

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

`plstart(device, nx, ny)`

Alternative to **plstar** for initializing the plotting package. The device name keyword for the desired output device must be supplied as an argument. The device keywords are the same as those printed out by **plstar**. If the requested device is not available, or if the input string is empty or begins with "?", the prompted startup of **plstar** is used. 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.

device (char *, input):
Device name (keyword) of the required output device. If NULL or if the first character is a "?", the normal (prompted) startup is used.

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

ny (PLINT, 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**.(also see **pllsty**)

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

space (PLINT *, 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 (PLFLT, input):
The distance of the left-hand edge of the viewport from the left-hand edge of the subpage in millimetres.

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

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

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

`plsxax(digmax, digits)`

Returns current values of the **digmax** and **digits** flags for the x axis. **digits** is updated after the plot is drawn, so so this routine should only be called *after* the call to **plbox** (or **plbox3**) is complete. See section 3.5 for more information.

digmax (PLINT, output):
Variable to set the maximum number of digits. If nonzero, the printed label will be switched to a floating point representation when the number of digits exceeds **digmax**.

digits (PLINT, input):
Field digits value. Currently, changing its value here has no effect since it is set only by **plbox** or **plbox3**. However, the user may obtain its value after a call to **plbox** by calling **plgxax**.

`plsyax(digmax, digits)`

Identical to **plsxax**, except that arguments are flags for y axis. See the description of **plsxax** for more detail.

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

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

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

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

`plszax(digmax, digits)`

Identical to **plsxax**, except that arguments are flags for z axis. See the description of **plsxax** for more detail.

`pltext()`

Sets an interactive device to text mode, used in conjunction with **plgra** to allow graphics and text to be interspersed. On a device which supports separate text and graphics windows, this command causes control to be switched to the text window. This can be useful for printing diagnostic messages or getting user input, which would otherwise interfere with the plots. The user *must* switch back to the graphics window before issuing plot commands, as the text (or console) device will probably become quite confused otherwise. If already in text mode, this command is ignored. It is also ignored on devices which only support a single window or use a different method for shifting focus. See also **plgra**.

`plvasp(aspect)`

Sets the viewport so that the ratio of the length of the y axis to that of the x axis is equal to `aspect`.

`aspect` (PLFLT, input):
Ratio of length of y axis to length of x axis.

`plvpas(xmin, xmax, ymin, ymax, aspect)`

Device-independent routine for setting up the viewport. The viewport is chosen to be the largest with the given aspect ratio that fits within the specified region (in terms of normalized subpage coordinates). This routine is functionally equivalent to `plvpor` when a "natural" aspect ratio (0.0) is chosen. Unlike `plvasp`, this routine reserves no extra space at the edges for labels.

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

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

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

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

`aspect` (PLFLT, input):
Ratio of length of y axis to length of x axis.

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

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

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

`ymax` (PLFLT, 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.

`plw3d(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 3.11 on page 19 for a more complete description of three-dimensional plotting.

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

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

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

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

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

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

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

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

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

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

`az` (PLFLT, 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.

`plwid(width)`

Sets the pen width.

`width` (PLINT, input):
The desired pen width. The pen width must be between 1 and a device dependent maximum value.

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

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

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

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

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

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

Appendix A

Programmer interface

A.1 C language

The argument types given in this manual (PLFLT and PLINT) are typedefs for the actual argument type. A PLINT is actually a type `long` and should not be changed. A PLFLT can be either a `float` or `double`; this choice is made when the package is installed and on a Unix system (for example) may result in a PLPLOT library named `libplplotf.a` in single precision and `libplplotd.a` in double precision.

These and other constants used by PLPLOT are defined in the main header file `plplot.h`, which must be included by the user program. This file also contains all of the function prototypes, machine dependent defines, and redefinition of the C-language bindings that conflict with the Fortran names (more on this later). `plplot.h` obtains its values for PLFLT, PLINT, and PLARGS (a macro for conditionally generating prototype argument lists) from FLOAT (typedef), INT (typedef), and PROTO (macro), respectively. The latter are defined in the file `chdr.h`. The user is encouraged to use FLOAT, INT, and PROTO in his/her own code, and modify `chdr.h` according to taste. It is not actually necessary to declare variables as FLOAT and INT except when they are pointers, as automatic conversion to the right type will otherwise occur (if using a Standard C compiler; else K&R style automatic promotion will occur). The only code in `plplot.h` that directly depends on these settings is as follows:

```
#include "chdr.h"

/* change from chdr.h conventions to plplot ones */

typedef FLOAT PLFLT;
typedef INT   PLINT;
#define PLARGS(a) PROTO(a)
```

PLPLOT is capable of being compiled with Standard C (ANSI) mode on or off. This is toggled via the macro PLSTDC, and set automatically if __STDC__ is defined. If PLSTDC is defined, all functions are prototyped as allowed under Standard C, and arguments passed exactly as specified in the prototype. If PLSTDC is not defined, however, function prototypes are turned off and K&R automatic argument promotion will occur, e.g. `float` → `double`, `int` → `long`. There is no middle ground! A PLPLOT library built with PLSTDC defined will not work (in general) with a program built with PLSTDC undefined, and vice versa. It is possible in principle to build a library that will work under both Standard C and K&R compilers simultaneously (i.e. by duplicating the K&R promotion with the Standard C prototype), but this seems to violate the spirit of the C standard and can be confusing. Eventually we will drop support for non-standard C compilers but for now have adopted this compromise.

In summary, PLPLOT will work using either a Standard or non-standard C compiler, provided that you :

1. Include the PLPLOT main header file `plplot.h`.
2. Make sure all pointer arguments are of the correct type (the compiler should warn you if you forget, so don't worry, be happy).
3. Do not link a code compiled with PLSTDC defined to a PLPLOT library compiled with PLSTDC undefined, or vice versa.
4. Use prototypes whenever possible to reduce type errors.

Note that some Standard C compilers will give warnings when converting a constant function argument to whatever is required by the prototype. These warnings can be ignored.

The one additional complicating factor concerns the use of stub routines to interface with Fortran (see the following section for more explanation). On some systems, the Fortran & C namespaces are set up to clobber each other. More reasonable (from our viewpoint) is to agree on a standard map between namespaces, such as the appending of an underscore to Fortran routine names as is common on many Unix-like systems. The only case where the shared Fortran/C namespaces do any good is when passing a pointer to a like data type, which represents only a small fraction of the cases that need to be handled (which includes constant values passed on the stack, strings, and two-dimensional arrays).

There are several ways to deal with this situation, but the least messy from a user's perspective is to redefine those PLPLOT C function names which conflict with the Fortran-interface stub routines. The actual function names are the same as those described in this document, but with a "c_" prepended. These macro definitions appear in the `plplot.h` header file and are otherwise harmless. Therefore you can (and should) forget that most of the names are being redefined to avoid the conflict and simply adhere to the bindings as described in this manual. Codes written under old versions of PLPLOT (previous to 4.0) will require a recompile, however.

It is possible to compile the PLPLOT library so that the affected functions retain their documented names, by specifying `-DNOBRAINDEAD` on the compile line. Some reasons for doing this might be: (a) you don't want to include `plplot.h` into your program for whatever reason, or (b) you are debugging PLPLOT and wish to specify the documented names to the debugger. In fact this procedure will work fine under SUNOS, with no Fortran/C namespace collision. But on other systems a special compiler switch may be required to get it to work similarly, which is an extra burden on the user and best avoided.

For more information on calling PLPLOT from C, please see the 14 example C programs (`x01c.c` through `x14c.c`) distributed with PLPLOT.

A.2 Fortran language

As discussed in the preceding section, PLPLOT's integer representation is a PLINT and its floating point representation is a PLFLT. To the Fortran user, this most commonly translates to a type `integer` and type `real`, respectively. This is somewhat system dependent (and up to the installer of the package) so you should check the release notes to be sure, or just try it and see what happens.

Because the PLPLOT kernel is written in C, standard C syntax is used in the description of each PLPLOT function. Thus to understand this manual it is helpful to know a little about C, but fortunately the translation is very easy and can be summarized here. As an example, the routine `plline` call from C would look like:

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

while from Fortran would look like:

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

typically with `n` declared as type `integer` and `x`, `y` declared as type `real` (arrays in this case). Each C language type used in the text translates roughly as follows:

PLFLT	real
PLINT	integer
char *	character
PLFLT *	real or real array
PLFLT **	real array
"string"	'string'
array[0]	array(1)

In C there are two ways to pass a variable — by value (the default) or by reference (pointer), whereas only the latter is used by Fortran. Therefore when you see references in the text to *either* an ordinary argument or a pointer argument (e.g. `*data`), you simply use an ordinary Fortran variable or array name.

The PLPLOT library comes with a set of Fortran interface routines that allow the exact same call syntax (usually) regardless of whether calling from C or Fortran. These "stub" routines handle transforming the data from the normal Fortran representation to that typically used in C. This includes:

1. Variables passed by value instead of by reference.

   Fortran passes all subroutine arguments by reference, i.e. a pointer to the argument value is pushed on the stack. In C all values, except for arrays (including char arrays), are passed by value, i.e. the argument value itself is pushed on the stack. The stub routine converts the Fortran call by reference to a call by value. As an example, here is how the plpoin stub routine works. In your Fortran program you might have a call to plpoin that looks something like

   ```
       call plpoin(6,x,y,9)
   ```

   where x and y are arrays with 6 elements and you want to plot symbol 9. As strange as it seems (at least to C programmers) the constants 6 and 9 are passed by reference. This will actually call the following C stub routine (included in entirety)

   ```
   #include "plstubs.h"

   void
   PLPOIN(n, x, y, code)
   PLINT *n, *code;
   ```

```
PLFLT *x, *y;
{
    c_plpoin(*n, x, y, *code);
}
```

All this stub routine does is convert the number of points (`*n` and the symbol `*code` to call by value (i.e. pushes their value on the stack) and then calls the C plpoin library routine.

2. Get mapping between Fortran and C namespace right (system dependent).

   The external symbols (i.e. function and subroutine names) as you see them in your program often appear differently to the linker. For example, the Fortran routine names may be converted to uppercase or lowercase, and/or have an underscore appended or prepended. This translation is handled entirely via redefinition of the stub routine names, which are macros. There are several options for compiling PLPLOT that simplify getting the name translation right (see Appendix C for more info). In any case, once the name translation is established during installation, name translation is completely transparent to the user.

3. Translation of character string format from Fortran to C.

   Fortran character strings are passed differently than other quantities, in that a string descriptor is pushed on the stack along with the string address. C doesn't want the descriptor, it wants a NULL terminated string. For routines that handle strings two stub routines are necessary, one written in Fortran and one written in C. Your Fortran program calls the Fortran stub routine first. This stub converts the character string to a null terminated integer array and then calls the C stub routine. The C stub routine converts the integer array (type `long`) to the usual C string representation (which may be different, depending on whether your machine uses a big endian or little endian byte ordering; in any case the way it is done in PLPLOT is portable). See the **plmtex** stubs for an example of this.

   Note that the portion of a Fortran character string that exceeds 299 characters will not be plotted by the text routines (**plmtex** and **plptex**).

4. Multidimensional array arguments are changed from row-dominant to column-dominant ordering through use of a temporary array.

   In Fortran, arrays are always stored so that the first index increases most rapidly as one steps through memory. This is called "row-dominant" storage. In C, on the other hand, the first index increases *least* rapidly, i.e. "column-dominant" ordering. Thus, two dimensional arrays (e.g. as passed to the contour or surface plotting routines) passed into PLPLOT must be transposed in order to get the proper two-dimensional relationship to the world coordinates. This is handled in the C stub routines by dynamic memory allocation of a temporary array. This is then set equal to the transpose of the passed in array and passed to the appropriate PLPLOT routine. The overhead associated with this is normally not important but could be a factor if you are using very large 2d arrays.

This all seems a little messy, but is very user friendly. Fortran and C programmers can use the same basic interface to the library, which is a powerful plus for this method. The fact that stub routines are being used is completely transparent to the Fortran programmer.

For more information on calling PLPLOT from Fortran, please see the 13 example Fortran programs (`x01f.f` through `x13f.f`) distributed with PLPLOT.

Appendix B

Output devices

PLPLOT supports a variety of output devices, via a set of device drivers. Each driver is required to emulate a small set of low-level graphics primitives such as initialization, line draw and page advance, as well as be completely independent of the PLPLOT package as a whole. Thus a driver may be very simple, as in the case of the many black & white file drivers (tektronix, etc.). More complicated and/or color systems require a bit more effort by the driver, with the most effort required by an output device with a graphical user interface, including menus for screen dumps, palette manipulation, and so forth. At present only the Amiga drivers do the latter, although we plan to eventually offer a nice user interface on other systems as well (X-windows, OS/2). At present we aren't pursuing a Macintosh development effort due to a lack of time and expertise, but will assist anyone wanting to volunteer for the job.

A Computer Graphics Metafile driver is under development and may be finished by the time you read this (see the release notes to be sure). Note that if you always render to a PLPLOT metafile, you can always **plrender** them to new devices as they become available.

The list of available devices presented when starting PLPLOT (via **plstar**) is determined at compile-time. When installing PLPLOT you may wish to exclude devices not available on your system in order to reduce screen clutter. To include a specified device, simply define the appropriate macro constant when building PLPLOT (see the installation instructions for your system).

The device drivers for PLPLOT terminal output at present are given in table B.1, with drivers for file output given in table B.2. The driver for OS/2 PM is available separately. See the section on OS/2 in Appendix C for more details.

Device	keyword	driver file
Xterm Window	xterm	xterm.c
X-Window Screen	xwin	xwindow.c
DOS VGA Screen	vga	dosvga.c
Amiga Window/Screen	amiwn	amiwn.c
Tektronix Terminal	tekt	tektronx.c
DG300 Terminal	dg300	dg300.c

Table B.1: PLPLOT terminal output devices

Device	keyword	driver file
PLPLOT Native Meta-File	plmeta	plmeta.c
Tektronix File	tekf	tektronx.c
PostScript File	ps	pscript.c
LaserJet II Bitmap File (150 dpi)	ljii	ljii.c
XFig file	xfig	xfig.c
Amiga Printer	amipr	amipr.c
IFF file (Amiga)	iff	iff.c
Aegis Draw file (Amiga)	aegis	aegis.c
HP 7470 Plotter File (HPGL Cartridge Small Plotter)	hp7470	hp7470.c
HP 7580 Plotter File (Large Plotter)	hp7580	hp7580.c
Impress File	imp	impress.c

Table B.2: PLPLOT file output devices

B.1 Driver functions

A dispatch table is used to direct function calls to whatever driver is chosen at run-time. Below are listed the names of each entry in the generic dispatch table, defined in **dispatch.h**. The entries specific to each device (defined in **dispatch.c**) are typically named similarly but with "pl_" replaced by a string specific for that device (the logical order must be preserved, however). The dispatch table entries are :

- **pl_MenuStr** : Pointer to string that is printed in device menu.
- **pl_DevName** : A keyword (string) for device selection by name.
- **pl_init** : Initialize device. This routine may also prompt the user for certain device parameters or open a graphics file. Called only once to set things up.
- **pl_line** : Draws a line between two points.
- **pl_clear** : Clears screen or ejects page or closes file.
- **pl_page** : Set up for plotting on a new page. May also open a new a new graphics file.
- **pl_adv** : Advance to the next page. Equivalent to a **pl_clear** followed by a **pl_page** on most devices.
- **pl_tidy** : Tidy up. May close graphics file.
- **pl_color** : Change pen color.
- **pl_text** : Switch device to text mode.
- **pl_graph** : Switch device to graphics mode.

- `pl_width` : Set graphics pen width.
- `pl_esc` : Escape function for driver-specific commands.

It is recommended that when adding new functions to only a certain driver, the escape function be used. Otherwise it is necessary to add a null routine to all the other drivers to handle the new function.

For more information, see `dispatch.h` or `dispatch.c`.

B.2 PLPLOT metafiles and plrender

The PLPLOT metafile is a way to store and transport your graphical data for rendering at a later time or on a different system. A PLPLOT metafile is in binary format in order to speed access and keep storage costs reasonable. All data is stored in device-independent format (written as a stream of bytes); the resulting file is about as portable as a tektronix vector graphics file and only slightly larger.

Each PLPLOT metafile begins with a header string that identifies it as such, as well as the version number (year & letter) of the format since this may change in time. The utility for rendering the metafile, `plrender`, verifies that the input file is indeed a valid PLPLOT metafile, and that it "understands" the format the metafile is written in. `plrender` is part of the PLPLOT package and should be built at the time of building PLPLOT, and then put into your search path. It is capable of high speed rendering of the graphics file, especially if the output device can accept commands at a high rate (e.g. X windows).

The commands as written by the metafile driver at present are as follows:

- INITIALIZE
- CLOSE
- SWITCH_TO_TEXT
- SWITCH_TO_GRAPH
- CLEAR
- PAGE
- NEW_COLOR
- NEW_WIDTH
- LINE
- LINETO
- ESCAPE
- ADVANCE

Each command is written as a single byte, possibly followed by additional data bytes. The NEW_COLOR and NEW_WIDTH commands each write 2 data bytes, the LINETO command writes 4 data bytes, and the LINE command writes 8 data bytes. The most common instruction in the typical metafile will be the LINETO command, which draws a continuation of the previous line to the given point. This data encoding is not quite as efficient as the tektronix format, which uses 4 bytes instead of 5 here (1 command + 4 data), however the PLPLOT encoding is far simpler to implement and more robust. The ESCAPE function writes a second command character (opcode) followed by an arbitrary number of data bytes depending on the value of the opcode. Note that any data written must be in device independent form to maintain the transportability of the metafile (so floating point numbers are not allowed *per se*).

The short usage message for `plrender` is printed if one inputs insufficient or invalid arguments, and is as follows:

```
% plrender

usage:  plrender [-h] [-v] [-dev name] [-i name] [-o name] [-f] [-geo geom]
    [-b number] [-e number] [-p page] [-a aspect] [-ori orient] [-px number]
    [-py number] [-fam] [-fsiz size] [-fmem member] [-np] [filename]

Type plrender -h for a full description.
```

The longer usage message goes into more detail, and is as follows:

```
% plrender -h

usage:
        plrender [-options ...] [filename]

where options include:
    -h                  Print out this message
    -v                  Print out version info
    -dev name           Output device name
    -i name             Input filename
    -o name             Output filename, or X server to contact
    -f                  Filter option -- equivalent to "-i - -o -"
    -geo geom           X window size, in pixels (e.g. -geo 400x400)
    -b number           Beginning page number
    -e number           End page number
    -p page             Plot given page only
    -a aspect           Plot aspect ratio
    -ori orient         Plot orientation (0=landscape, 1=portrait)
    -px number          Plots per page in x
    -py number          Plots per page in y
    -fam                Create a family of output files
    -fsiz size          Output family file size in MB (e.g. -fsiz 1.0)
    -fmem member        Starting family member number on input [1]
    -np                 No pause between pages

All parameters must be white-space delimited.  If you omit the "-i" flag,
the filename parameter must come last.  Specifying "-" for the input or
output filename means use stdin or stdout, respectively.  Only one filename
parameter is recognized.  Not all options valid with all drivers.
Please see the man pages for more detail.
```

These are generally self explanatory (family files are explained below). Most of these options have default values, and for those that don't `plrender` will prompt the user. The `-px` and `-py` options are not so useful at present, because everything is scaled down by the specified factor — resulting in labels that are too small (future versions of `plrender` might allow changing the label size as well).

Additional options may be added in future releases.

B.3 Family file output

When sending PLPLOT to a file, the user has the option of generating a "family" of output files for certain output drivers (at present this is available for the plmeta, tektronix, and postscript drivers only). This can be valuable when generating a large amount of output, so as to not strain network or printer facilities by processing extremely large single files. Each family member file can be treated as a completely independent file. In addition, `plrender` has the ability to process a set of family member files as a single logical file.

To create a family file, one must simply call **`plsfam`** with the familying flag **`fam`** set to 1, and the desired maximum member size (in bytes) in **`bmax`**. If the current output driver does not support familying, there will be no effect. This call must be made *before* calling **`plstar`** or **`plstart`**.

If familying is enabled, the name given for the output file (in response to the **`plstar`** prompt or as a **`plstart`** argument) becomes the stem name for the family. Thus, if you request a plmeta output file with name **`test.plm`**, the files actually created will be **`test.plm.1`**, **`test.plm.2`**, and so on. A new file is automatically started once the byte limit for the current file is passed, but not until the next page break.

The **`plgfam`** routine can be used from within the user program to find out more about the graphics file being written. In particular, by periodically checking the number of the member file currently being written to, one can detect when a new member file is started. This information might be used in various ways; for example you could spawn a process to automatically plrender each metafile after it is closed (perhaps during a long simulation run) and send it off to be printed.

`plrender` has several options for dealing with family files. It can process a single member file (**`plrender test.plm.1`**) or the entire family if given only the stem name (**`plrender test.plm`**). It can also create family files on output, rendering to any device that supports familying, including another metafile if desired. The size of member files in this case is input through the argument list, and defaults to 1MB if unspecified (this may be changed during the PLPLOT installation, however). **`plrender`** can also create a single output file from a familied input metafile.

B.4 Interactive output devices

Here we shall discuss briefly some of the more common interactive output devices.

Many popular terminals or terminal emulators at present have a facility for switching between text and graphics "screens". This includes the xterm emulator under X-windows, vt100's with Retrographics, and numerous emulators for microcomputers which have a dual vt100/tek4010 emulation capability. On these devices, it is possible to switch between the text and graphics screens by surrounding your PLPLOT calls by calls to **`plgra()`** and **`pltext()`**. This will allow your diagnostic and informational code output to not interfere with your graphical output.

At present, only the xterm driver supports switching between text and graphics screens. The escape sequences as sent by the xterm driver are fairly standard, however, and have worked correctly on most other popular vt100/tek4010 emulators we've tried.

When using the xterm driver, hitting a RETURN will advance and clear the page. If indeed running from an xterm, you may resize, move, cover and uncover the window. The behavior of the X-window driver is quite different, however. First, it is much faster, as there is no tty-like handshaking going on. Second, a mouse click is used to advance and clear the page, rather than a RETURN. And last, the window cannot be resized or covered without losing the graphics inside (this will be improved in a later release). On the other hand, the Amiga screen driver and the OS/2 PM driver both support resizing and exposure with automatic refresh.

On a tektronix 4014 compatible device, you may preview tektronix output files via the **pltek** utility. **pltek** will let you step through the file interactively, skipping backward or forward if desired. The help message for **pltek** is as follows:

```
Usage: pltek filename
At the prompt, the following replies are recognized:

    h      Gives this help message.
    ?      As above.
    q      Quits program.
   <n>     Goes to the specified frame number (surprise!).
           If negative, will go back the specified number of frames.
 <Return>  Goes to the next frame (first frame, if initial reply).
```

The output device is switched to text mode before the prompt is given, which causes the prompt to go to the vt102 window under xterm and most vt100/tek4010 emulators.

Appendix C

System dependent usage and installation

Here are listed notes for PLPLOT usage and installation on various systems. The user must have access to the proper PLPLOT library(s), utilities (`plrender` and optionally `pltek`), and runtime access to the required font files (`plstnd4.fnt, plxtnd4.fnt`). It is best to supplement these notes with a site-dependent document outlining the exact procedure for using PLPLOT.

Eventually we may make available the PLPLOT library binaries for different machines via anonymous ftp, but for now users typically must build the package themselves unless the site adminstrator has done so already. Typically this will entail unpacking the .zoo or .tar PLPLOT source archive on your system, making appropriate changes to set up the makefile or script files, and then letting the computer take over. A complete build of PLPLOT will typically take only a few minutes on a fast workstation. Systems that are currently on the "officially" supported list should pose few difficulties (we hope). If you find that your system and/or compiler require changes to the makefile and/or header files, feel free to contact us about including these in the next PLPLOT release.

C.1 Unix-type systems

C.1.1 Linking

Linking your program with the PLPLOT library on a Unix type system can be done using something similar to (this is system specific):

```
cc  -o main main.c libplplotf.a -lm -lX11     or
f77 -o main main.f libplplotf.a -lm -lX11
```

for the single precision library (`-lplplotd` for double). You can also link using the `-l` option to the linker, i.e.:

```
cc  -o main main.c -lplplotf -lm -lX11     or
f77 -o main main.f -lplplotf -lm -lX11
```

however in this case you must be very careful to get the order of libraries on the command line right (libraries to be searched last should be placed at the end of the command line). In the latter case the library (`libplplotf.a`) must be in the search path used by the linker or specified by the `-L` option.

The utilities **plrender** and **pltek** should be in your search path. Some good places to put these include `/usr/local/bin/` or `/usr/local/plplot/`.

The PLPLOT font files must be where the library can find them. The current directory is always searched for the fonts first, followed by a system-dependent (and perhaps site-dependent) search path. This can be modified from the makefile, but is structured so that it won't often need to be modified. Note that directory names must be defined with the trailing slash, if non-null. The default search order for Unix-type systems is as follows:

```
current directory
$(HOME)/lib/
$(PLFONTS)
PLFONTDEV1        (/usr/local/lib/)
PLFONTDEV2        (/usr/local/lib/plplot/
PLFONTDEV3        (/usr/local/plplot/)
```

This is will not be a concern to the user if the fonts are located correctly during installation.

C.1.2 Installation

To achieve a degree of flexibility and system independence for the PLPLOT metafile, it is written in the macro language m4. System dependencies are resolved conditionally via m4 macros. Using this method, only one file (**makefile.m4**) is required for all systems with a SystemV-compatible **make**. This enables us to easily support many different systems.

Here's how to make everything conveniently on a unix-like system. First, if you run into problems, you should read **sys/unix/makefile.m4** carefully, as it has much additional information inside. If your system is not already supported, you must add system specific defines analogous to the existing ones in **makefile.m4**. Then:

```
% cd tmp
% cp ../sys/unix/makefile.m4 .
% m4 -D<sys> makefile.m4 >makefile
```

where ⟨sys⟩ is an abbreviation for your system name, e.g. SUNOS, UNICOS, AIX, etc. To get the double precision library for your system, specify `-DDOUBLE` on the **m4** command line (note: Fortran double precision may not be supported on your system using this method; check the release notes to be sure). At this point, you may wish to edit **makefile** to change whatever settings are required *for your site only*. For example, you may wish to edit the device list to include only those devices which you actually have access to.

Then, while in the **tmp** directory:

```
% make links
```

sets up soft links to all the files you will ever need.

```
% make [libs]
```

will make the PLPLOT library(s) (**libs** is the first target and therefore optional), whereas

```
% make everything
```

makes the main library, fonts, **plrender**, and **pltek**, for the default precision (probably single on a workstation). This is just an abbreviation for typing **% make libs fonts plrender pltek**. To make one of the example programs,

```
% make x01c
```

for the first C demo, or

```
% make x01f
```

for the first Fortran demo (other example programs are similar). To make all the C program or Fortran program demos, specify `cdemos` or `fdemos` as the target.

Finally, you must move the PLPLOT library, fonts, and utilities to a more permanent location, for example:

```
% mv ../lib/*.lib /usr/local/plplot
% mv *.fnt *.idx *.lkp /usr/local/plplot
% mv plrender pltek /usr/local/plplot
```

if you are installing into system directories, or

```
% mv ../lib/*.lib ~/lib
% mv *.fnt *.idx *.lkp ~/lib
% mv plrender pltek ~/bin
```

if you are installing in your directory tree only (note under SUNOS you must run **ranlib** on libraries after moving them). You should then test the library again using one of the example programs to ensure that everything works perfectly. After you are finished, you may delete everything in the tmp directory.

C.2 VMS

C.2.1 Linking

On VMS, the PLPLOT library is split into several parts due to limitations of the VMS linker when used with the VAXC compiler. These are:

```
libplplotf1.obj
libplplotf2.obj
```

for the single precision library. You will need to link with these as well as **sys$library:vaxcrtl/lib**.

You will also need to make a symbol definition to allow **plrender** to run as it is meant to, if not already set up (note: **pltek** currently doesn't work on VMS). This is done in the following way:

```
$ plrender :== $public:plrender.exe
```

if **public** is the logical name for the directory containing **plrender**. Then **plrender** will run the same as on Unix systems.

The PLPLOT font files must be where the library can find them. The font locating code looks in the following places for the fonts:

- current directory
- lib:
- sys$login: (home directory)

- PLFONTDIR (a makefile macro, set when PLPLOT is installed)

This is will not be a concern to the user if the fonts are located in PLFONTDIR as set during installation (defaults to **sys$sysroot:[sysfont.plplot]**).

One difficulty the user may encounter when running PLPLOT under VMS arises from ideosyncracies of the VAX C compiler. Namely, binary files as created by a VAX C program are always of record format STREAM_LF, while the customary binary file transfer protocols (ftp, kermit) insist on "Fixed length 512 byte records". Note that DECNET copy works on all file record formats and does not suffer from this problem.

Thus, any file created by PLPLOT under VMS must be transformed to the more standard VMS binary file format (fixed length records) before doing anything with it. This includes printing or transfer to another system. Contrawise, when transferring a PLPLOT metafile to VMS, you must first convert it to STREAM_LF format before rendering it with **plrender**. There are several small, public domain utilities available to do this conversion, for example the **bilf** utility that comes with the **zoo** archive program (by Rahul Dhesi). A copy of **bilf.c** is distributed with PLPLOT in the sys/vms directory in case you do not have it already.

C.2.2 Installation

On VMS the build is a bit complicated although a makefile is provided (using Todd Aven's MAKE/VMS, a PD make-like utility). If you do not have MAKE/VMS installed on your system, you can either: (a) get a copy, (b) rewrite the given makefile to work with DEC's MMS, or (c) get a copy of the object files from someone else. For further information, see **[.sys.vms]makefile**.

Note: the X window driver is NOT included. I do not know what libraries to link with or how common they are on vaxen in general (I've used at least one vax which does not have any X libraries on it). If you come up with a way to deal with this (portably), feel free to send your changes.

C.3 Amiga

C.3.1 Linking

With the Lattice C compiler, linking to PLPLOT is accomplished by something like (this depends on the compiler options you have used when creating the library):

```
lc -o main -Lm+plplot.lib main.c
```

The PLPLOT font files must be where the library can find them. The current directory is always searched for the fonts first, followed by **fonts:plplot/**, and **plfonts:**, in that order, with one additional place that can be specified from the makefile. If you put them in **fonts:plplot/** you should never get a requester asking you to mount **plfonts:**.

For a color requester PLPLOT tries in a couple of places. The palette program used should open in the top-most screen (note that **sys:prefs/palette** shipped with the 2.0 operating system does not do this). Under 2.0, **sys:tools/colors** is used (standard). Under 1.3, it is recommended that you make avalable the **palette** program that comes with the WorkBench 1.3 enhancer package. PLPLOT looks for this program in **tools:** and if it's not there it looks in **sys:tools**, so you'll want to assign **tools:** to the directory in which the palette program resides.

C.3.2 Installation

The build for the Amiga is very similar to the one for Unix. There are two extra utilities you need for this to proceed flawlessly:

- `make` (PD version, ported from mod.sources posting)
- `m4` (also PD)

`m4` is used to convert the master makefile.m4 into a makefile suited for your system. This handles system dependencies, compiler dependencies, and choice of math library. From the `plplot/tmp` directory, you can create the makefile via:

```
% copy /sys/unix/makefile.m4 ""
% m4 -DAMIGA <makefile.m4 >makefile
```

The default is to use IEEE single precision math, with the library **`libs:mathieeedoubbas.library`** located at run-time. By contrast, if you want to use the Motorola Fast Floating Point library, you would build the makefile via:

```
% m4 -DAMIGA -DMATH=FFP <makefile.m4 >makefile
```

Eventually there will be switches for compiler as well (right now it is set up for SAS/C; if someone makes the changes for their favorite compiler, please send me a copy).

Then, while in the tmp directory:

```
% make links
```

copies all of the source files to the current directory. It is much more efficient to use links instead if you are using the 2.0 operating system. There is a script provided (**`makelinks`**) for SKsh users that sets up links to all the necessary files.

All the other targets (except for those dealing with the Fortran interface) are identical to that under a Unix-like system; please see Section C.1 for more detail. The Fortran interface is not supported on the Amiga at present, as the vendors of Fortran compilers currently use a custom object file format which prevents linkage of Fortran programs with C programs.

C.3.3 IFF and printer driver

The IFF driver will prompt for resolution, page size, filename and orientation unless **`plspage`** and **`plsfile`** or **`plsfnam`** are used.

The printer driver creates a black and white graph on your preferences supported printer. It uses the preferences selected density and page size limits and ignores most of the other stuff. If you have selected "ignore" in the page size limit options then a full page graph is produced. The other options operate as described in the Amiga documentation. At present, only "ignore" and "bounded" produce aspect ratio correct plots (usually unimportant unless x and y must have the same scaling e.g. for pie charts or polar plots). You can get very high quality plots with this driver. A full page, high resolution plot requires a lot of memory, however (an 8"x10" 300 dpi plot requires $(8 * 300) * (10 * 300)/8 = 900000$ bytes). You can use the page limits to reduce the memory requirements and still get high quality output.

C.3.4 HP Plotter (or PLT: device)

The PLT: device is an HP plotter compatible device handler written by Jim Miller and Rich Champeaux. It is freely redistributable, but it is not included with this package. It gives high quality output like the preferences printer driver but with full preferences support. Also, it usually requires less memory (for full page plots) than the preferences driver, but is slower. Highly recommended if you have a color printer or are memory strapped. The PLT: device accepts virtually all of the standard HP-GL commands (probably the HP7470 is the best PLPLOT device driver to use).

C.3.5 Amiga window driver

Written by Tony Richardson. This provides a normal window with standard intuition gadgets. You can resize the window even while the program is plotting in the window (see the "Redraw Enabled" section below). If you are making several graphs on separate pages, use the close gadget to advance from one "page" to the next.

The PLPLOT menu selections are:

- Save Configuration
  Saves current window configuration (window size, screen type, screen depth, colors, resolution, etc.). The configuration is saved in `s:Plplot.def` (only 54 bytes).

- Reset
  Resets the window to the configuration in `s:Plplot.def` (or to a default config if this file doesn't exist).

- Maintain Plot Aspect
  If this is checked the plot aspect ratio is maintained as the window is resized (handy for polar plots, etc.). Default is unchecked in which case x and y are stretched independently to fit in the window.

- Redraw Enabled
  If this is checked, then the graphics commands are buffered in `t:plplot.dat`. This file is used to redraw the plot when required. It's status is checked only when a new graph is started. The buffer is also used to create the "Full Page" prints under the "Print" selection.

- Select Screen Type
  A submenu allows the user to select either "Workbench" or "Custom".

- Print
  There are three submenu options here. The "Bitmap Dump" does just that (with full preferences support). The output is pretty jagged, but you can play around with the preferences stuff (scaling, smoothing, etc.) to improve things. The other two submenus are "Full Page (Landscape)" and "Full Page (Portrait)". This uses the graphics buffer file (see "Redraw Enabled" above) to create graphics output similar to the preferences driver. However the aspect ratio can not be maintained. Same preferences options are used as in preferences driver.

- Save Bitmap as IFF File
  Self explanatory. You can use this to save your images and then touch them up (do area fills, etc.) with your favorite paint program.

- Screen Format (this menu only appears on the custom screen).
  Here you may select "Interlaced", "High Resolution", "Number of Colors", or "Set Color Palette".

C.4 OS/2

At the time of this writing (Fall '91), the current release of OS/2 is version 1.3. Consequently the current implementation of PLPLOT is 16-bit in nature, and was developed using Microsoft 16 bit compilers. Like all the rest of the industry, this author is anxiously awaiting the arrival of OS/2 2.0 with its accompanying 32-bit programming model. So, rest assured that there will be a port of PLPLOT to OS/2 2.0 with all possible speed. For the time being PLPLOT does work with OS/2 1.3, and has been used with OS/2 1.2 as well. This author has no experience with OS/2 versions prior to version 1.2, so no guarantees can be made on that.

See the README file in `sys\os2` for additional information not covered herein.

C.4.1 Compiler Support

The OS/2 implementation of PLPLOT was developed using Microsoft C 6.0 and Microsoft Fortran 5.0 (these compilers were used for the DOS port as well). The majority of the code is of course independent of any vendor specific compiler peculiarities, as evidenced by the wide range of supported platforms. However, in the case of OS/2 and DOS there is one notable exception. Specifically, the way Microsoft chose to implement string argument passing in Fortran 5.0. The details are rather involved, so will not be given here. Suffice it to say that it was necessary to write a special set of Fortran language interface stubs just for use with Microsoft Fortran 5. If you wish to use some other compiler, you may very well find that the Unix stubs work better. It is genuinely hard to imagine how anyone could write a compiler in such a way that would make string handling any more difficult than Microsoft did in Fortran 5.0.

Further note that the supplied makefile is written with the syntax of Microsoft NMAKE version 1.11. Please do NOT attempt to use versions of Microsoft NMAKE prior to 1.11, as both 1.0 and 1.1 have been known to dramatically fail, and even to trash source files (through the misapplication, of default rules). It should not be difficult to transform the provided makefile for use with other vendor's make facilities. If you do so, please send in the changes so they can be provided to other users in future releases of this software.

C.4.2 Installation

Compiling the libraries is not too much work if you have the compilers listed above. Just get into the `plplot\tmp` directory, and issue the following commands:

```
copy ..\sys\os2\makefile .
nmake links
nmake
nmake fonts
nmake plrender
nmake cdemos fdemos
```

The `fonts` target bears special mention. It compiles the programs which create the fonts, runs them, and copies the resulting font files to `\lib` on the current drive. You should make sure you have such a directory before you begin this process, or else change the makefile.

Once done building the software, you'll probably want to copy the files plplot.lib and plstub.lib to someplace like \lib and the plrender utility to someplace in your path.

For programs to run, they need access to the fonts. This is most easily accomplished by setting the plfonts environment variable. If you leave the fonts in the default place, the command would be:

```
set plfonts=d:\lib\
```

where d is the drive where you'll be keeping the fonts.

C.4.3 Linking

Once the libraries are built, linking with them is not too difficult at all.

```
cl /AL main.c plplot.lib
fl /AL prog.for plplot.lib plstub.lib
```

You will of course need to have the various compiler and linker recognized environment variables correctly specified. A sample file plplot\sys\os2\plconfig.cmd is provided to show how I do this. Here is the important portion of the file:

```
set  path=d:\c6.0\binp;d:\c6.0\binb;c:\f5.0\bin;%syspath%
set  path=%path%d:\util;c:\bndutil;d:\cmd

set   lib=d:\c6.0\lib;c:\f5.0\lib
set   include=d:\c6.0\include;c:\f5.0\include
set helpfiles=d:\c6.0\help\*.hlp
set    cl= /FPi87 /Lp
set    fl= /FPi87 /link /NOE /NOD:llibfor7 /NOD:llibc7 llibc7p llibf7pc

set   plfonts=d:\lib\
```

whwere syspath is an environment variable containing all the directories needed for the system to operate normally. For descriptions of each of the various command options, see the compiler reference materials. Note that PLPLOT is compiled using the large memory model. Note also that to use Microsoft Fortran with PLPLOT you will need to have the C compatible Fortran libraries built.

C.4.4 Presentation Manager Support

Providing PLPLOT graphics under OS/2 PM proved to be a demanding project. The basic problem is that in order for an OS/2 program to interact with PM, it must be event driven and must do all its work in response to messages which are sent to it by the PM. In other words, it is simply not possible with the current version of OS/2 (1.3 at the time of this writing) to write a program which operates in the normal procedural paradigm, and which makes calls to OS/2 services to draw things on the screen—that's just not the way it works. Rather a program which uses PM to draw windows and images must set up a message queue, and then go into an infinite loop in which it obtains messages from the queue and processes them. These messages are sent to the program by PM, and tell it to do things like "resize thyself", "redraw thyself", "redraw this subsection of window such and such", and so forth. This style of programming is basically the exact inverse of how normal scientific and technical programs are constructed. Furthermore, OS/2 PM applications cannot write to `stdout`. These restrictions conspire to make it very difficult to generate visual graphics from a "normal" program when run on OS/2. (The same is true of MS Windows, but due to the notorious instability of that environment, no effort has been made to solve the problem in that context–so far).

In spite of the substantial difficulties involved, an output capability for OS/2 PM has been developed which allows the user of PLPLOT to be totally unconcerned with the inner workings of a native PM code. That is, the user/programmer may write programs which call plplot functions, just as s/he would on any other platform, and it is possible to view the output in a window. `stdout` is also made available, as one would expect, so that in fact no special programming whatsoever is required to make a "normal" PLPLOT program work on OS/2.

Due to the substantial effort which was required to provide a transparent and high quality graphics output capability on OS/2, this PM specific code (and documentation) is not being provided under the same distribution terms as the rest of the PLPLOT package. Rather a small fee of $15 for individual users is required. Note, the free distribution of PLPLOT will run on OS/2—it just won't come with the PM driver.

To obtain the add-on package which will allow PLPLOT programs to generate output for OS/2 Presentation Manager, individual users should mail $15. to:

Physical Dynamics
P.O. Box 8556
Austin, Texas 78713

Companies or institutions with multiple users may obtain site licenses for the following rates:

First 100 users, $15 each.
Next 100 users, $10 each.
Additional users, $5 each.

Texas residents add 8% sales tax. When you send in your money, be sure to specify your preferred delivery mechanism. Options are: postal mailing of the floppy of your choice, and internet e-mail of uuencoded ZOO, ZIP, etc. Just state your preference. For those who request e-mail delivery, please include a postal address in case there is any problem with the networks. Companies requesting mulitple user site licenses should include a contact.

In brief, the capabilities of the OS/2 PM driver are:

- Windowed output. Like the X Windows driver.
- Image redraws when the window is exposed or resized. This is the most important way in which the OS/2 PM driver is superior to the X Windows driver.
- Optional automatic frame advance. The other major point of functionality which the X driver doesn't have (yet).
- Color selection compatible with the X driver.
- Minimizable/Maximizable, and preserves window contents (just a special case of item 2 above).

If you choose not to obtain the OS/2 PM option from Physical Dynamics, you can certainly still use the free version of PLPLOT on OS/2. The included metafile output option in PLPLOT allows you to generate a PLPLOT native metafile from any OS/2 PLPLOT application. You can then switch to a DOS box and view the metafile using the **plrender** utility. You will need to build a DOS version of **plrender** once, and store it in a place accessible to your DOS path. In fact, the PLPLOT metafile driver was originally invented specifically to allow viewing output in the DOS box from programs run in OS/2 native mode, prior to the development of the OS/2 PM driver.

Be sure to check for README files in the **plplot\sys\os2** directory for any additional information not available at the time of preparation of this manual.

C.5 MS-DOS

The DOS implementation of PLPLOT is based on the OS/2 version. The main difference being that the DOS version has a VGA driver. The VGA driver uses calls to the Microsoft graphics functions supplied with C 6.0 and QC 2.0 and above. If you wish to use a different compiler which does not support the Microsft graphics functions, you will need to make a new driver, presumably based on the code structure of the existing Microsoft specific driver. This is not difficult however, so should not be perceived as a significant obstacle. Send in the new driver and they'll be incorporated into future versions for everyone's benefit.

In particular, the compilers used to develop the DOS version are the same as for the OS/2 version: Microsft C 6.0 and Fortran 5.0. Further, the installation procedure is identical, except get the makefile from **plplot\sys\dos**. Note that it is not possible to make a bound version of PLPLOT or of **plrender** since the DOS VGA driver contains code which cannot be bound.

See the README file in **sys\dos** for information not contained herein.

C.6 DOS 386

PLPLOT can also be built using the DOS 386 port of GNU CC by DJ Delorie. This version of of GNU CC generates code which requires a 386, and it uses a DOS extender to provide virtual memory. Using this compiler it is possible to write programs which are much much larger than will fit into the computer's memory. Details for how to obtain this compiler and build PLPLOT are provided in the file **plplot\sys\dos386\readme.1st**. Basically the procedure is just like that for DOS and OS/2.

One advantage of the DOS 386 version of PLPLOT over the straight DOS version, is that the DOS 386 version has a much improved display driver. Specifically, the DOS 386 driver understands how to operate most SVGA video cards, and will display in 256 colors on such cards. Consequently this driver is able to provide support for **plrgb** function, whereas the straight DOS driver currently does not.

See the **readme.1st** file for more information.

Figure Captions

The figures are all output from the PLPLOT Fortran example programs. These are practically identical to the example programs written in C, except for the addition of a polar contour plot in example program 9.

1. Example program 1 output showing different styles of line plots.
2. Example program 2 output showing color plots (not so impressive in black and white).
3. Example program 3 output showing polar plotting.
4. Example program 4 output showing logarithmic plotting and multiple scales per plot.
5. Example program 5 output showing histograms.
6. Example program 6 output showing the standard font.
7. Example program 7 output showing the extended font.
8. Example program 8 output showing 3-d plotting.
9. Example program 9 output showing contour plotting.
10. Example program 10 output showing absolute plotting.
11. Example program 11 output showing mesh plotting.
12. Example program 12 output showing histogram plotting with shading.
13. Example program 13 output showing a pie chart.

Bibliography

[1] M. J. LeBrun and T. Tajima. Particle simulation in curvilinear coordinate systems. Technical Report 374, Institute for Fusion Studies, University of Texas at Austin, 1989.

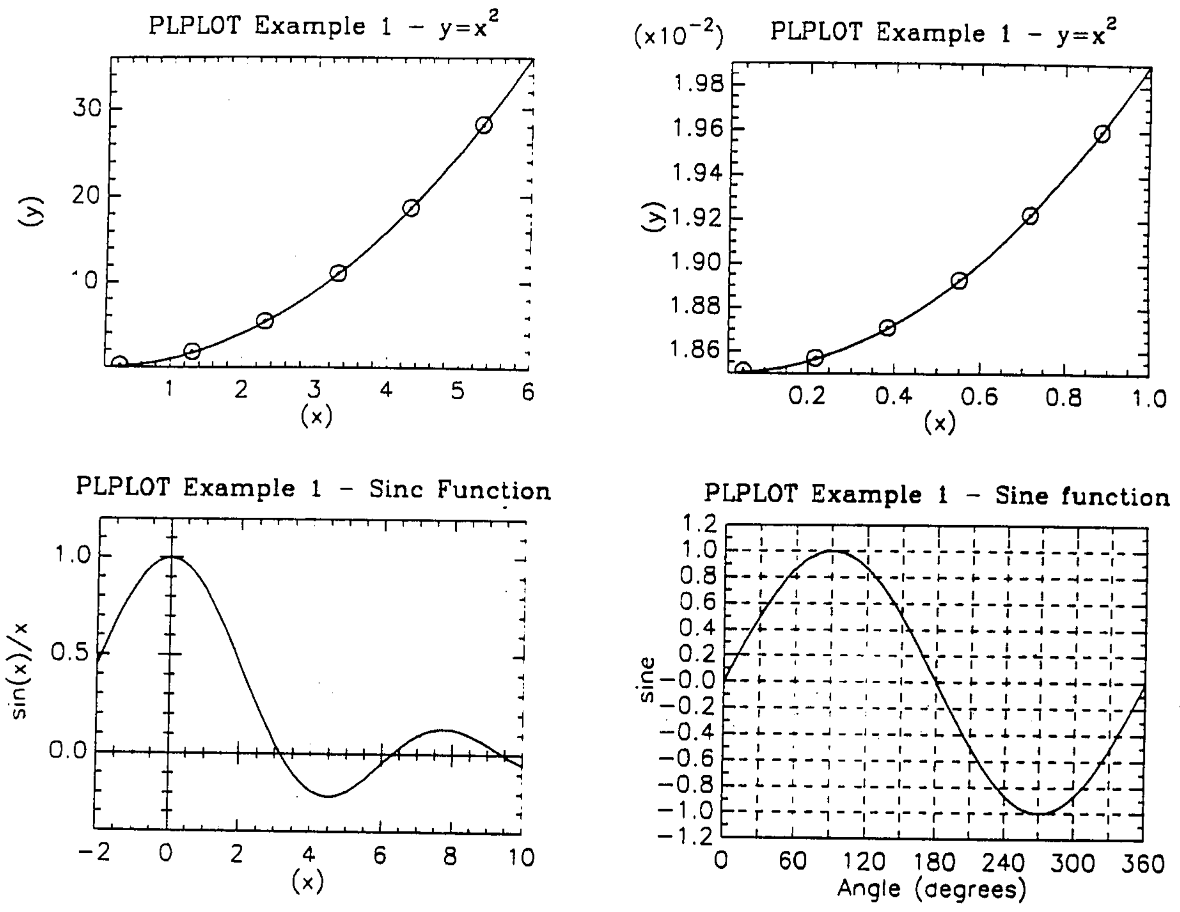

PLPLOT Example 1 - $y=x^2$
(y)
(x)
$(\times10^{-2})$
PLPLOT Example 1 - $y=x^2$
PLPLOT Example 1 - Sinc Function
sin(x)/x
PLPLOT Example 1 - Sine function
sine
Angle (degrees)

Fig. 1

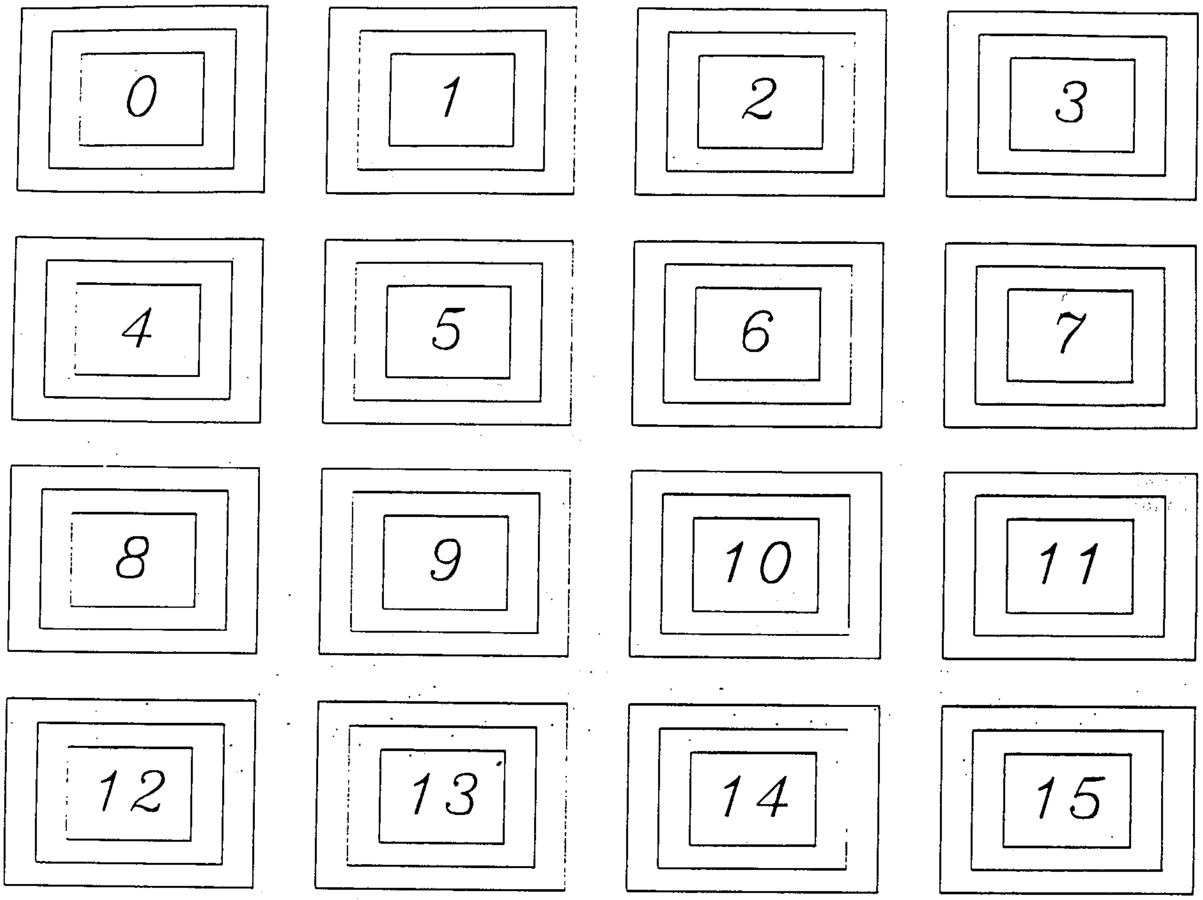


Fig. 2

PLPLOT Example 3 – $r(\theta)=\sin 5\theta$

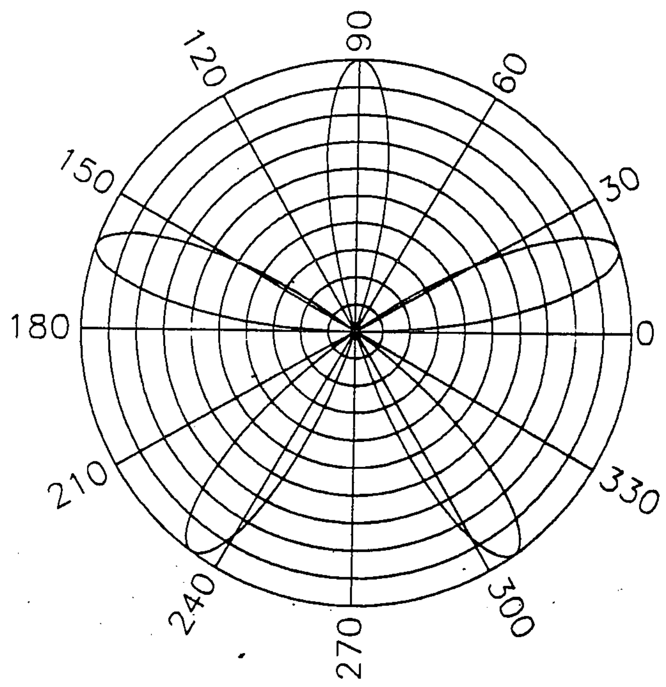
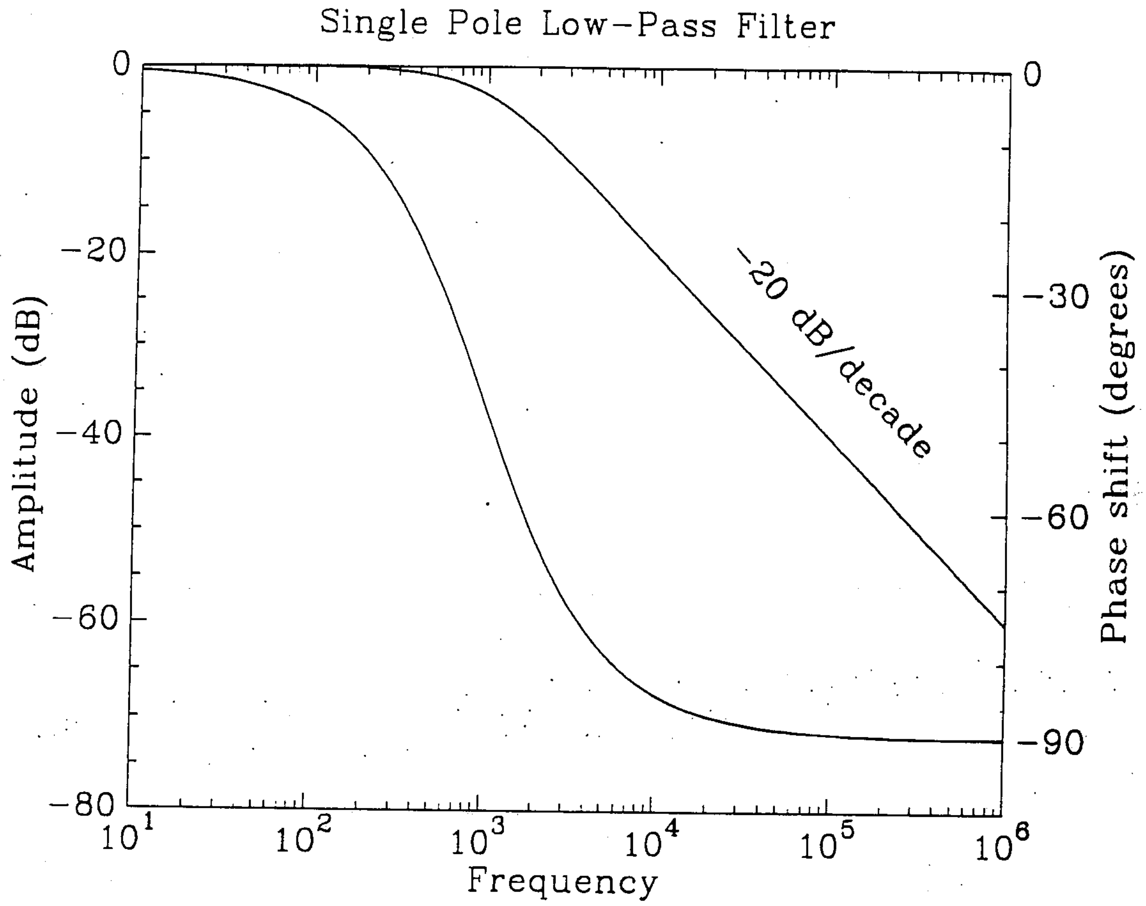


Fig. 3



Single Pole Low−Pass Filter
Amplitude (dB)
0
−20
−40
−60
−80
Phase shift (degrees)
0
−30
−60
−90
−20 dB/decade
Frequency
10^1
10^2
10^3
10^4
10^5
10^6

Fig. 4

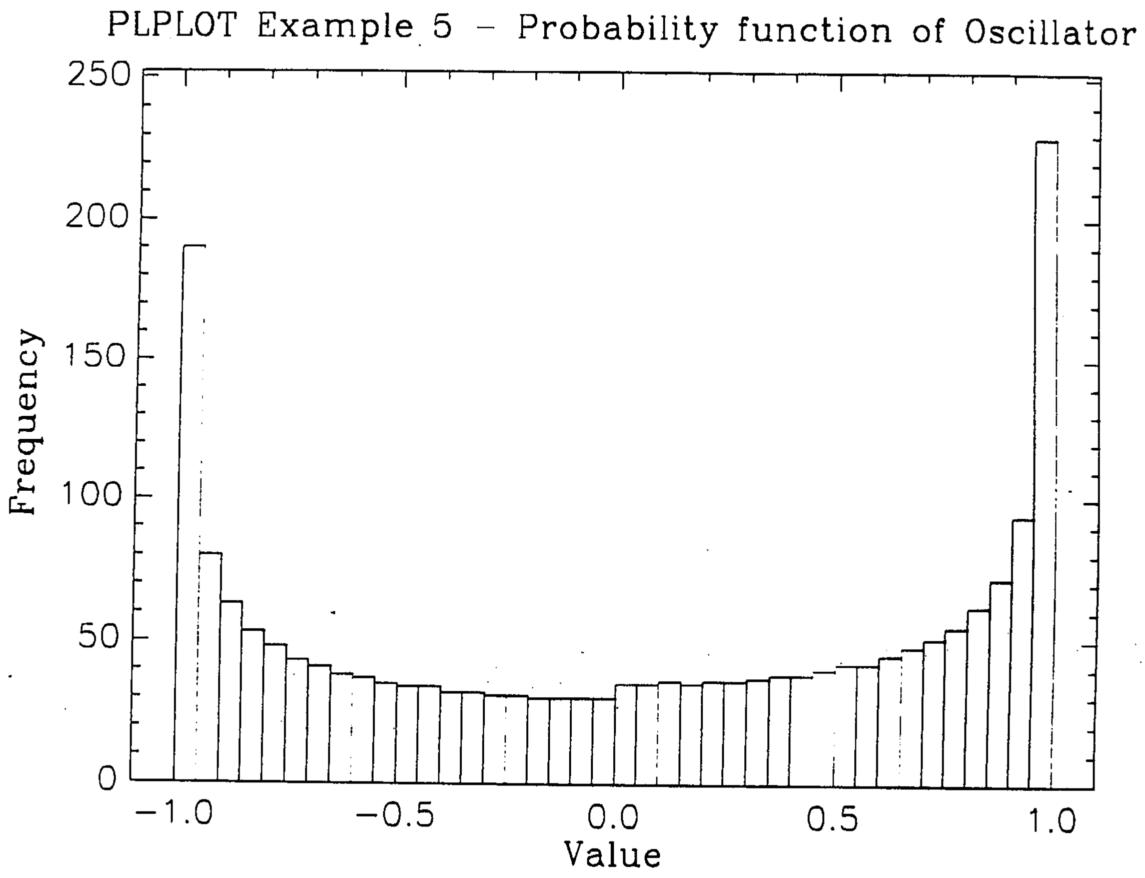

PLPLOT Example 5 - Probability function of Oscillator
250
200
150
100
50
0
Frequency
-1.0
-0.5
0.0
0.5
1.0
Value

Fig. 5

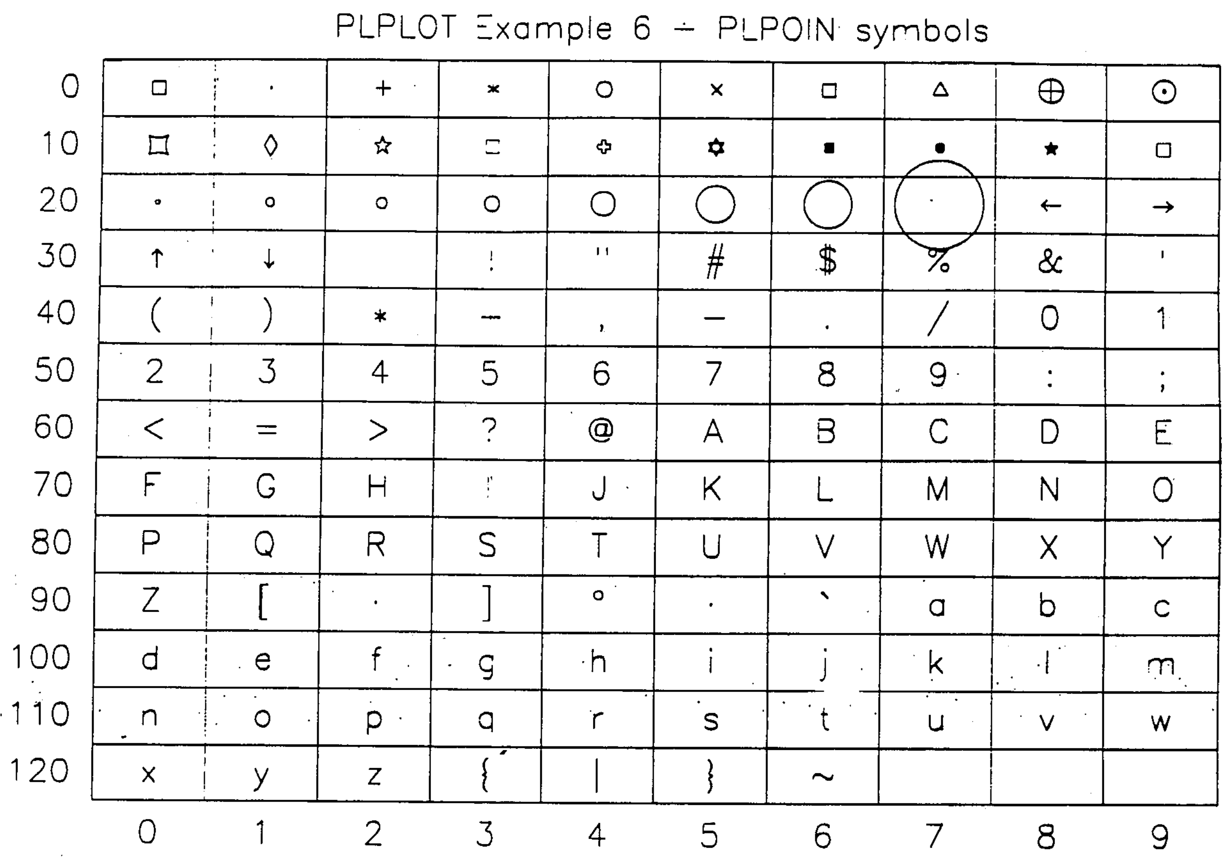


Fig. 6

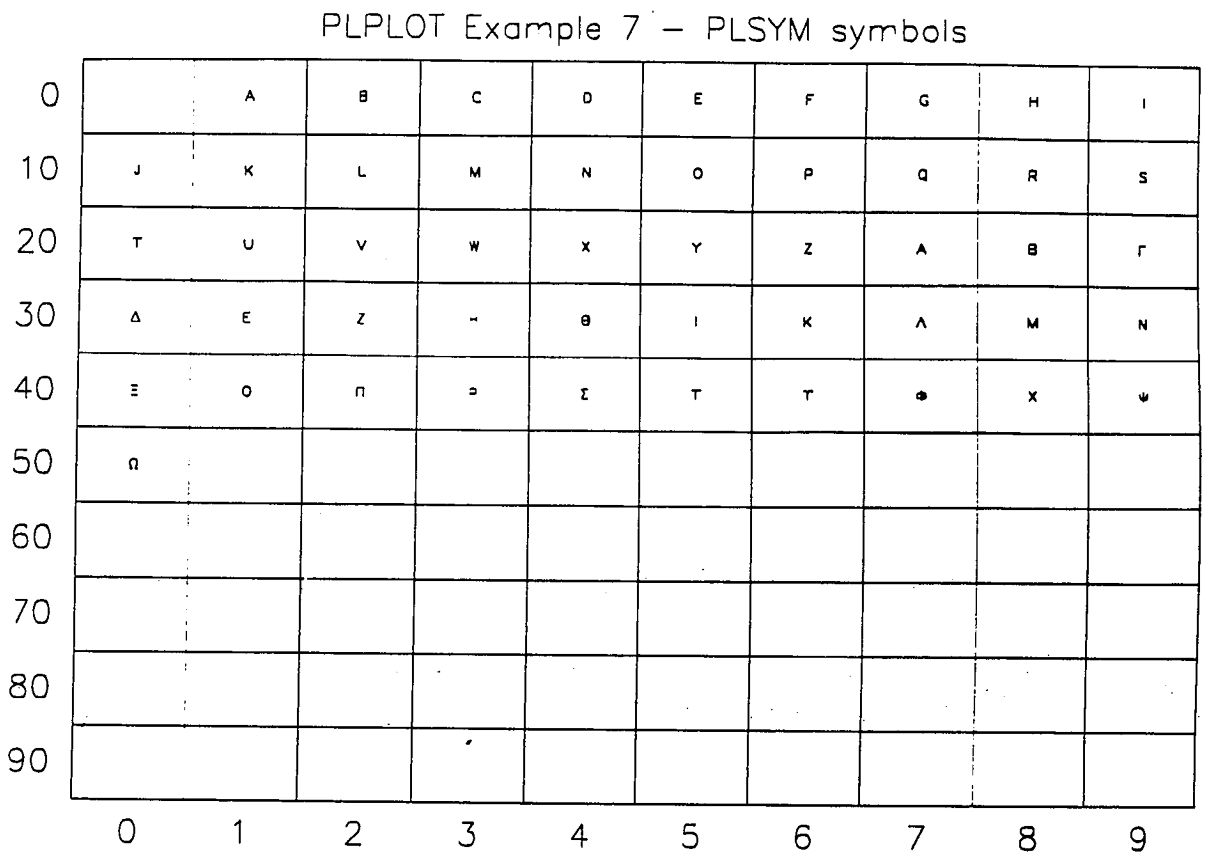


Fig. 7.1

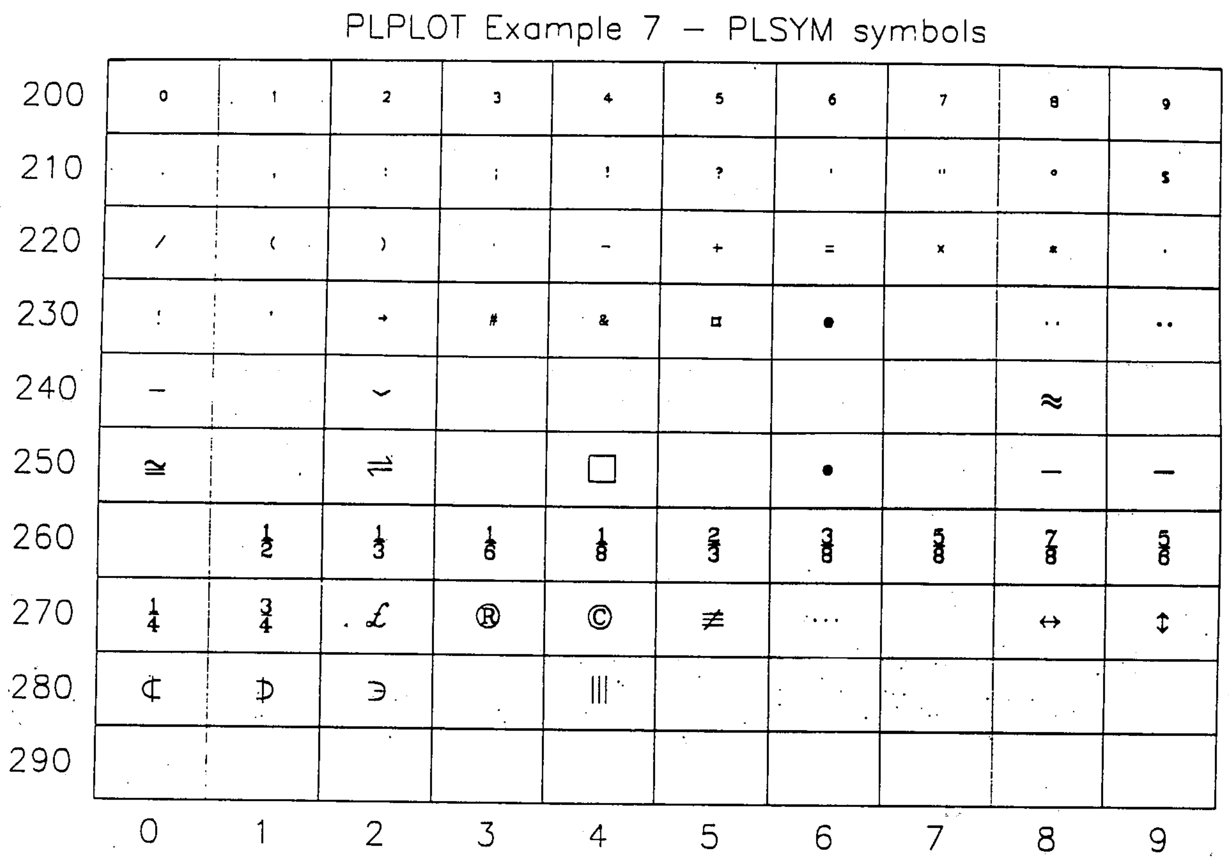
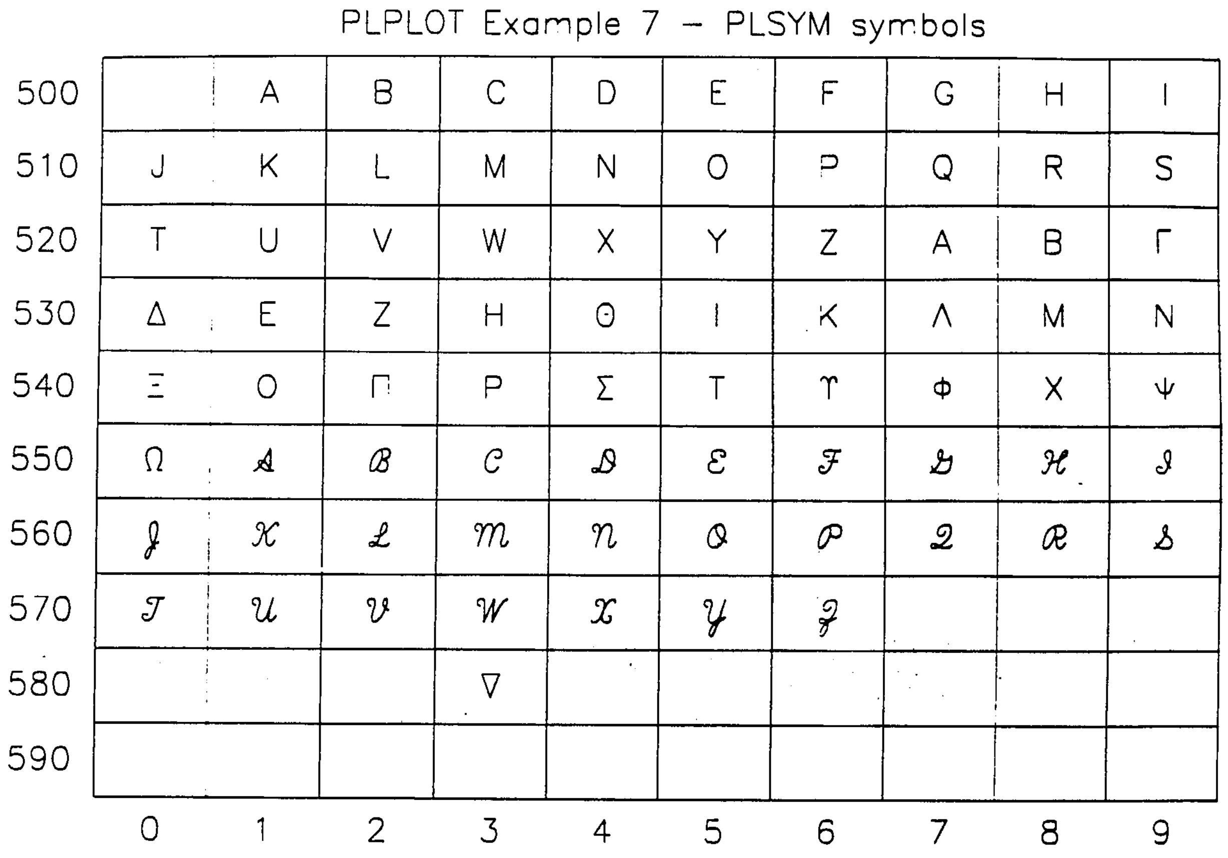
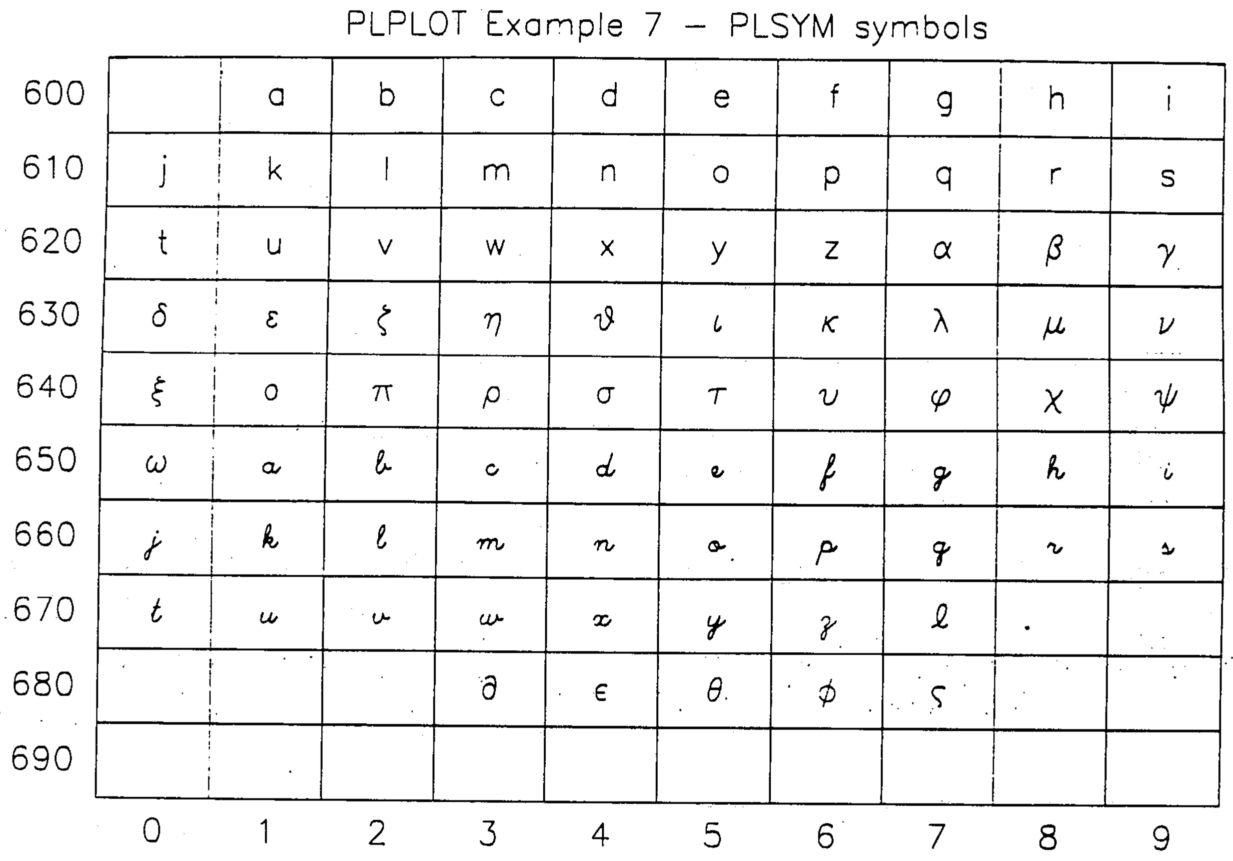


Fig. 7.2


PLPLOT Example 7 - PLSYM symbols

500		A	B	C	D	E	F	G	H	I
510	J	K	L	M	N	O	P	Q	R	S
520	T	U	V	W	X	Y	Z	A	B	Γ
530	Δ	E	Z	H	Θ	I	K	Λ	M	N
540	Ξ	O	Π	P	Σ	T	ϒ	Φ	X	Ψ
550	Ω	𝒜	ℬ	𝒞	𝒟	ℰ	ℱ	𝒢	ℋ	ℐ
560	𝒥	𝒦	ℒ	ℳ	𝒩	𝒪	𝒫	𝒬	ℛ	𝒮
570	𝒯	𝒰	𝒱	𝒲	𝒳	𝒴	𝒵			
580				∇						
590										
	0	1	2	3	4	5	6	7	8	9

Fig. 7.3



PLPLOT Example 7 – PLSYM symbols

600		a	b	c	d	e	f	g	h	i
610	j	k	l	m	n	o	p	q	r	s
620	t	u	v	w	x	y	z	α	β	γ
630	δ	ε	ζ	η	ϑ	ι	κ	λ	μ	ν
640	ξ	o	π	ρ	σ	τ	υ	φ	χ	ψ
650	ω	𝒶	𝒷	𝒸	𝒹	ℯ	𝒻	ℊ	𝒽	𝒾
660	𝒿	𝓀	𝓁	𝓂	𝓃	ℴ	𝓅	𝓆	𝓇	𝓈
670	𝓉	𝓊	𝓋	𝓌	𝓍	𝓎	𝓏	ℓ	.	
680				∂	ϵ	θ	ϕ	ς		
690										
	0	1	2	3	4	5	6	7	8	9

Fig. 7.4

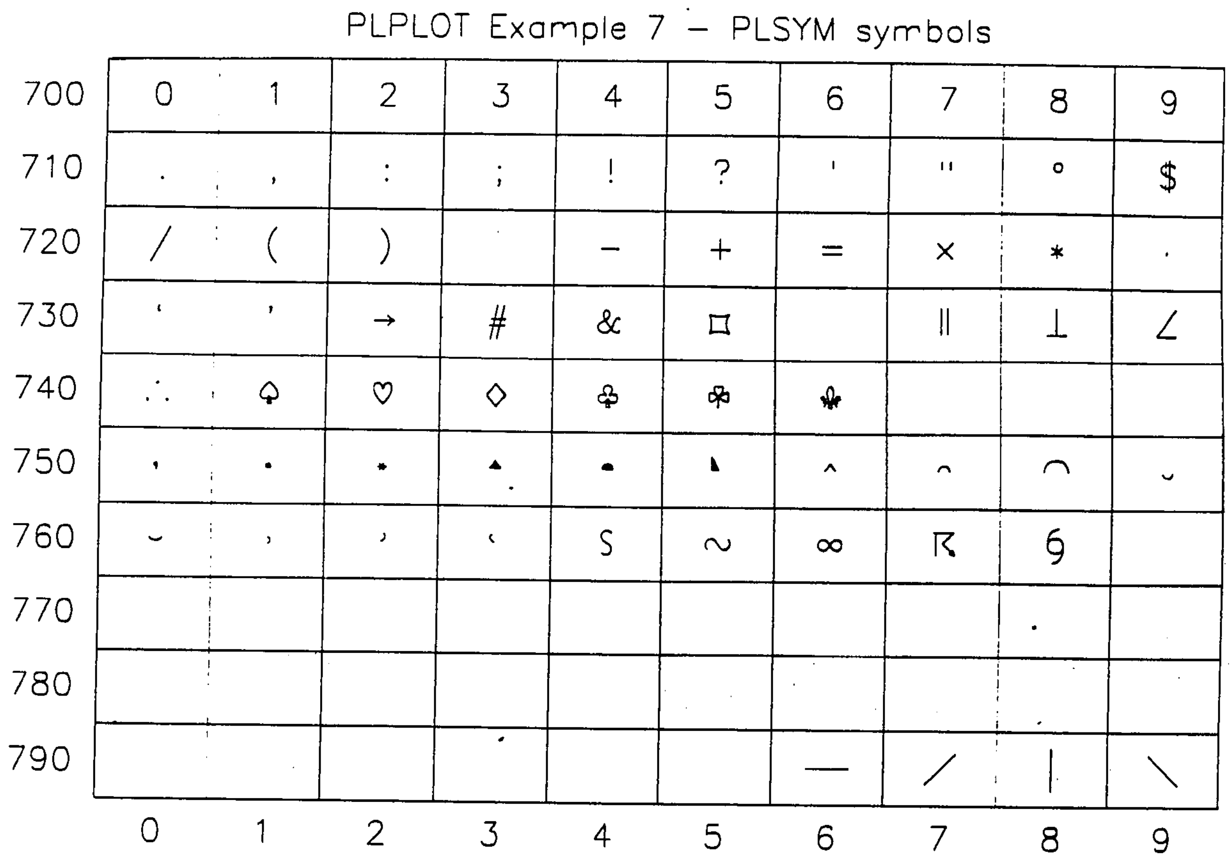


Fig. 7.5

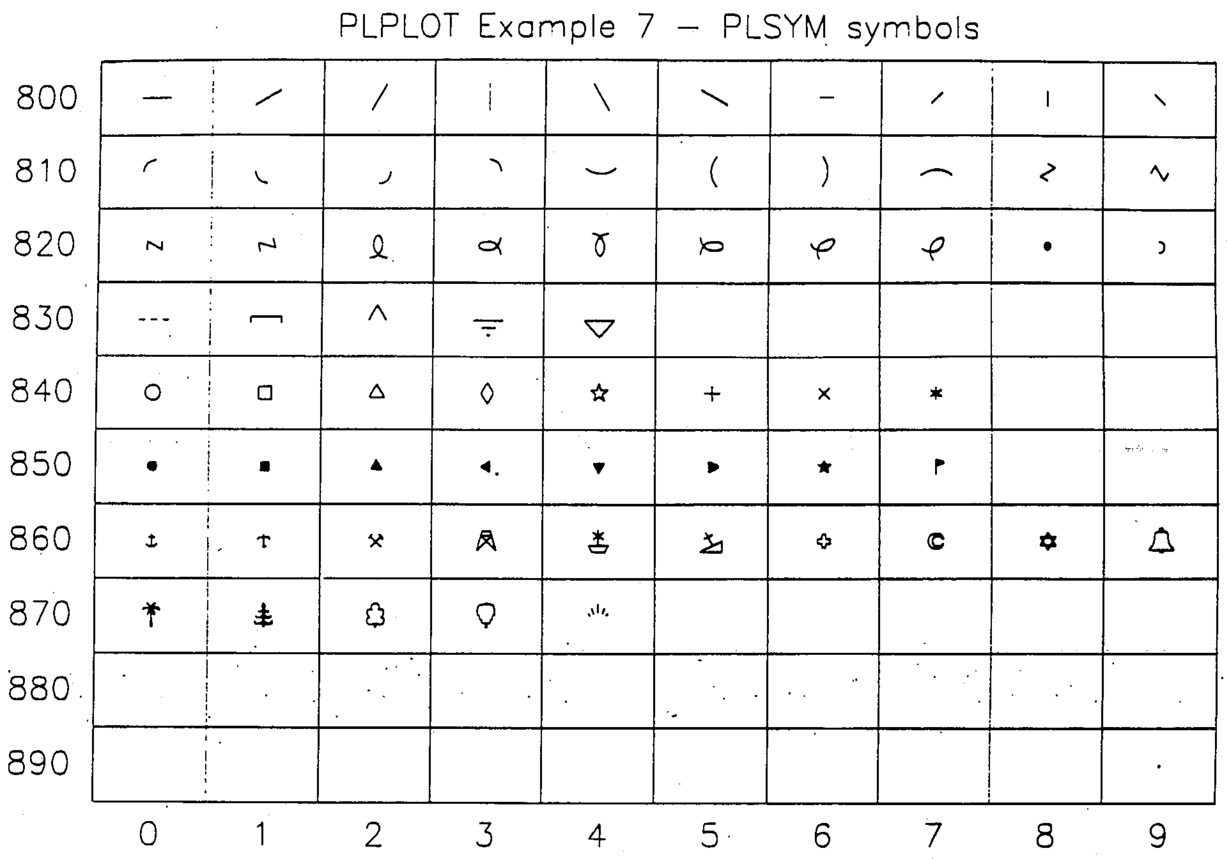

PLPLOT Example 7 — PLSYM symbols
800
810
820
830
840
850
860
870
880
890
0
1
2
3
4
5
6
7
8
9

Fig. 7.6

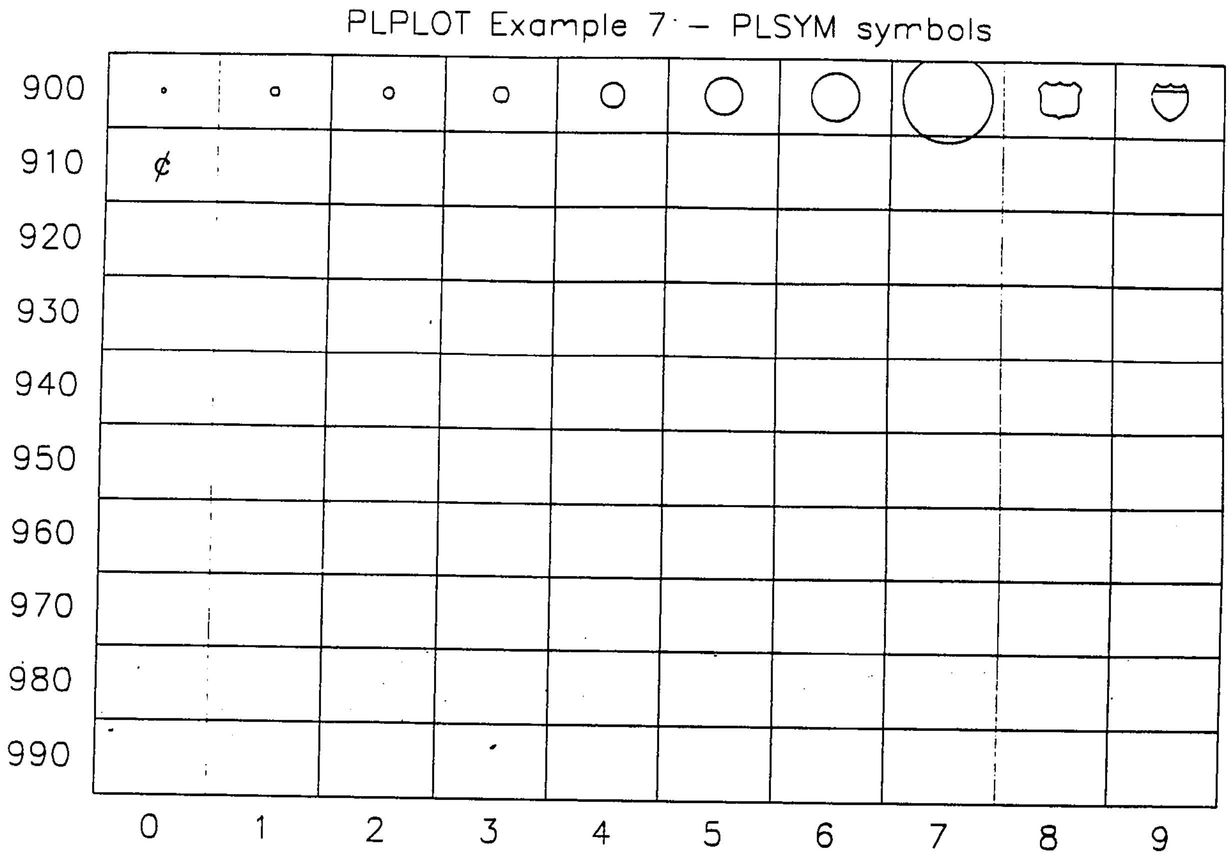
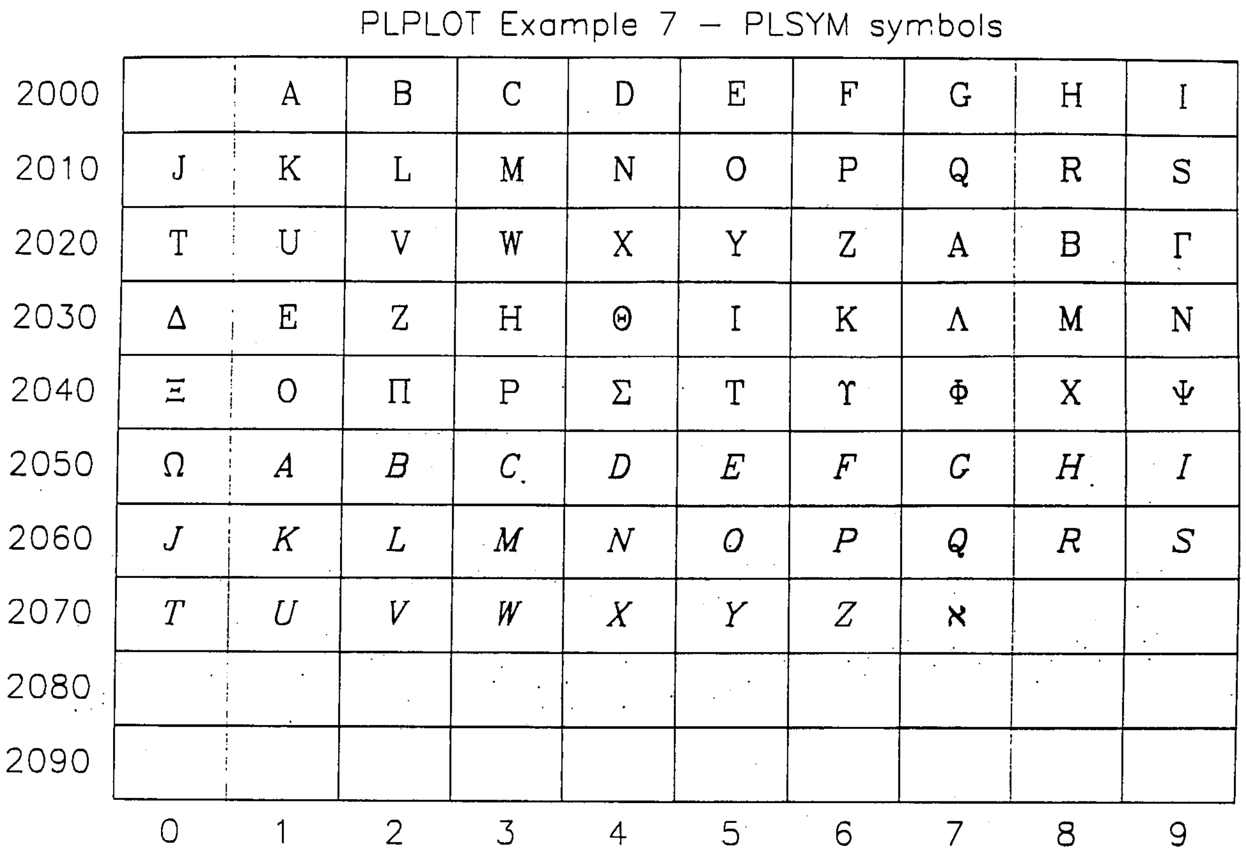

PLPLOT Example 7 – PLSYM symbols
900
910
920
930
940
950
960
970
980
990
0
1
2
3
4
5
6
7
8
9

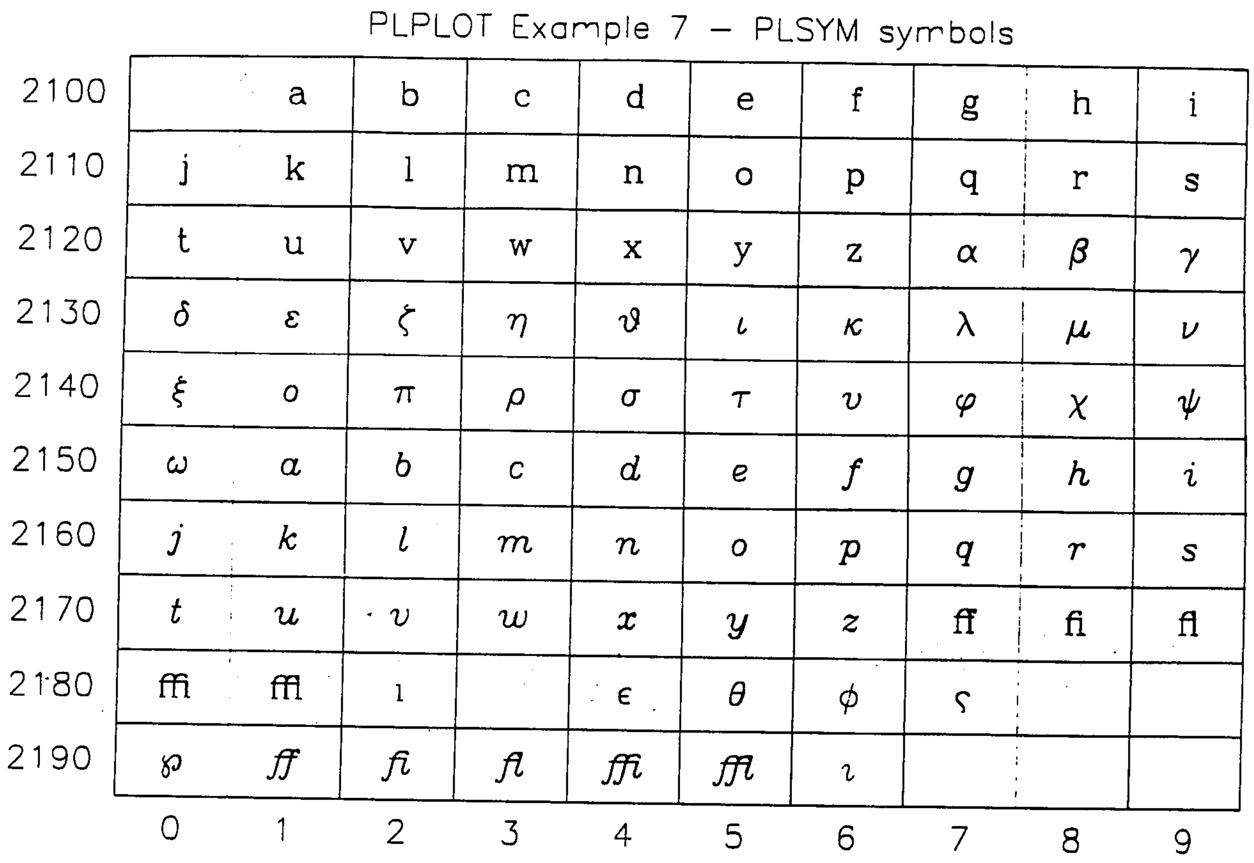
Fig. 7.7



PLPLOT Example 7 – PLSYM symbols

	0	1	2	3	4	5	6	7	8	9
2000		A	B	C	D	E	F	G	H	I
2010	J	K	L	M	N	O	P	Q	R	S
2020	T	U	V	W	X	Y	Z	Α	Β	Γ
2030	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν
2040	Ξ	Ο	Π	Ρ	Σ	Τ	ϒ	Φ	Χ	Ψ
2050	Ω	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*
2060	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*
2070	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	ℵ		
2080										
2090										

Fig. 7.8



PLPLOT Example 7 – PLSYM symbols

2100		a	b	c	d	e	f	g	h	i
2110	j	k	l	m	n	o	p	q	r	s
2120	t	u	v	w	x	y	z	α	β	γ
2130	δ	ε	ζ	η	ϑ	ι	κ	λ	μ	ν
2140	ξ	ο	π	ρ	σ	τ	υ	φ	χ	ψ
2150	ω	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*
2160	*j*	*k*	*l*	*m*	*n*	*o*	*p*	*q*	*r*	*s*
2170	*t*	*u*	*v*	*w*	*x*	*y*	*z*	ff	fi	fl
2180	ffi	ffl	ı		ϵ	θ	ϕ	ς		
2190	℘	*ff*	*fi*	*fl*	*ffi*	*ffl*	*ı*			
	0	1	2	3	4	5	6	7	8	9

Fig. 7.9

PLPLOT Example 7 – PLSYM symbols

2200	0	1	2	3	4	5	6	7	8	9
2210	.	,	:	;	!	?	′	″	°	*
2220	/	(	)	[	]	{	}	⟨	⟩	|
2230	‖	−	+	±	∓	×	·	÷	=	≠
2240	≡	<	>	≦	≧	∝	∼	ˆ	´	`
2250	˘	’	‘	‘	’	√	⊂	∪	⊃	∩
2260	∈	→	↑	←	↓	∂	∇	√	∫	∮
2270	∞	%	&	@	$	#	§	†	‡	∃
2280		☉	☿	♀	⊕	♂	♃	♄	♅	♆
2290	♇	☾	☄	✳	☊	☋	↗	↙	↖	↘
	0	1	2	3	4	5	6	7	8	9

Fig. 7.10

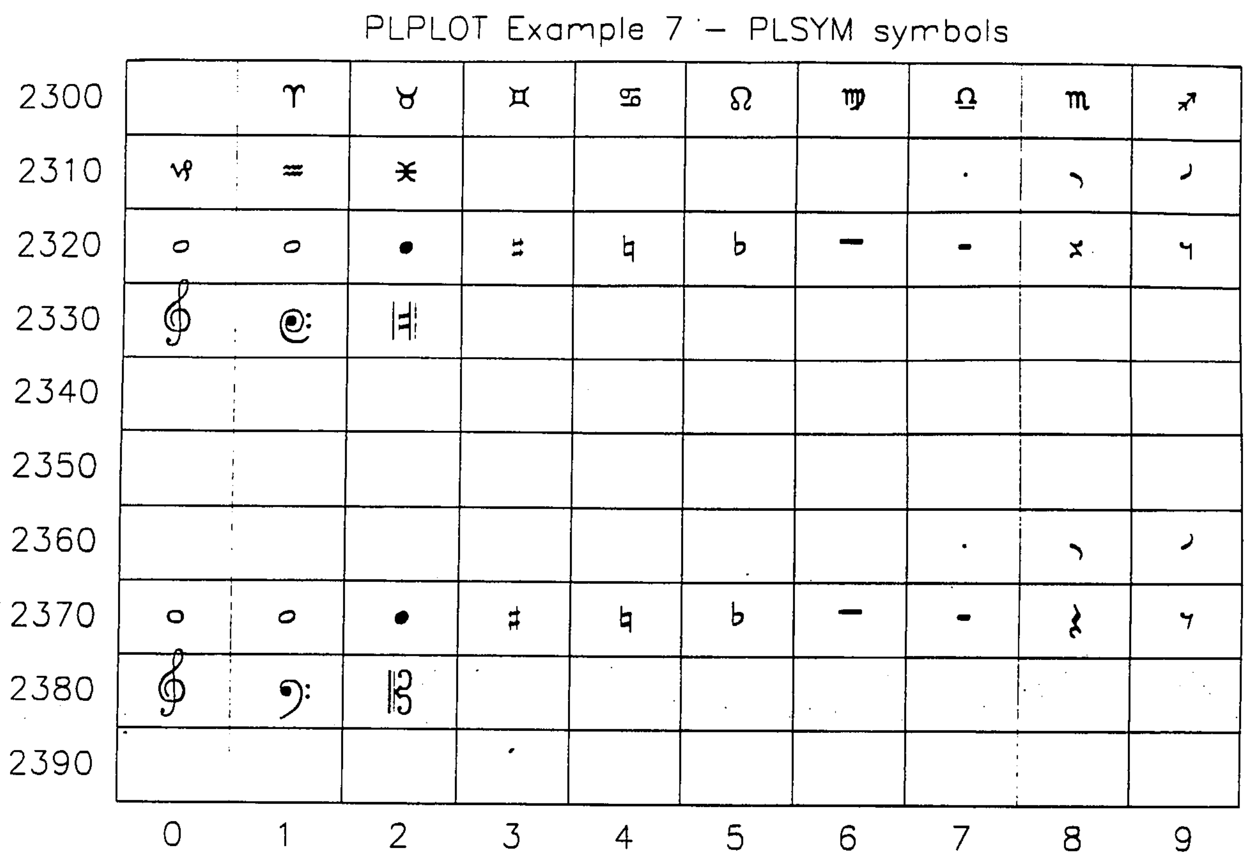
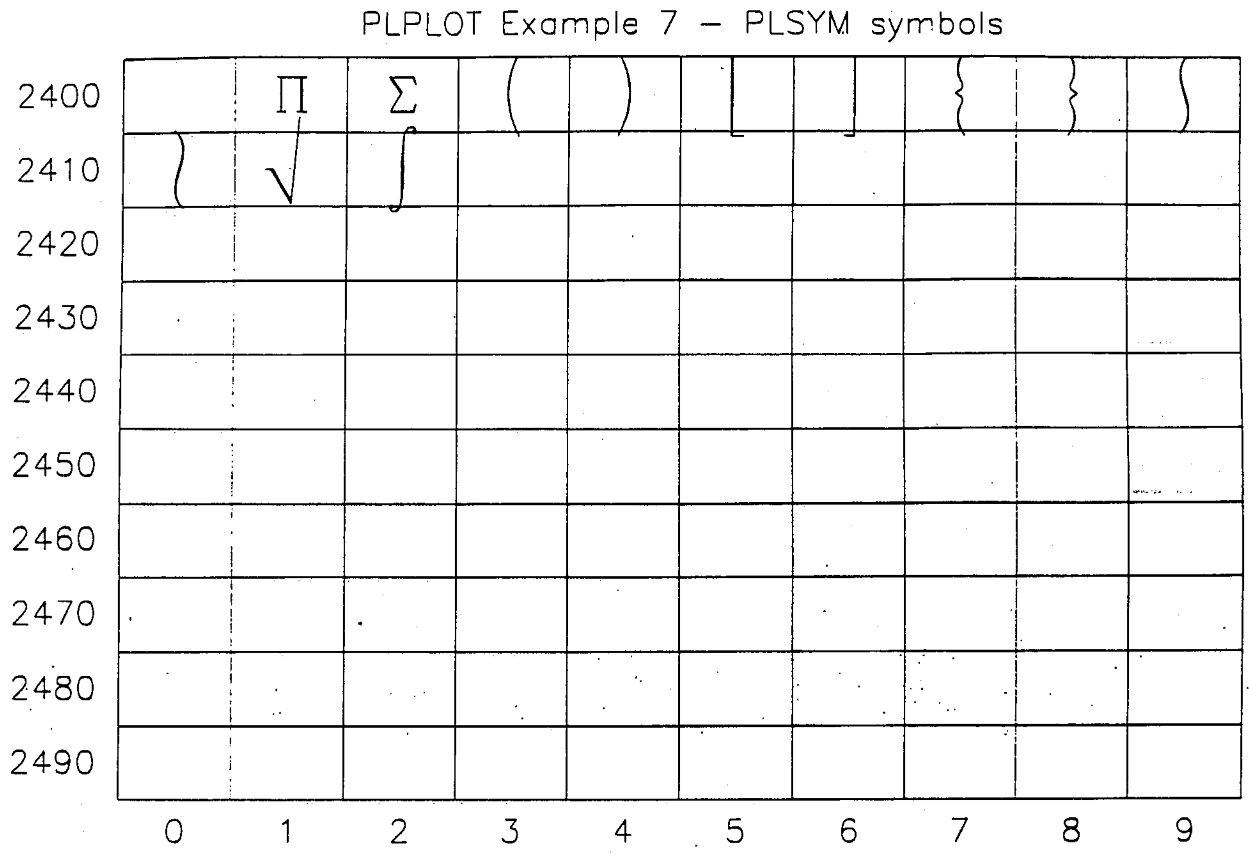

PLPLOT Example 7 – PLSYM symbols
2300
2310
2320
2330
2340
2350
2360
2370
2380
2390
0
1
2
3
4
5
6
7
8
9

Fig. 7.11



PLPLOT Example 7 - PLSYM symbols
2400
2410
2420
2430
2440
2450
2460
2470
2480
2490
0
1
2
3
4
5
6
7
8
9

Fig. 7.12

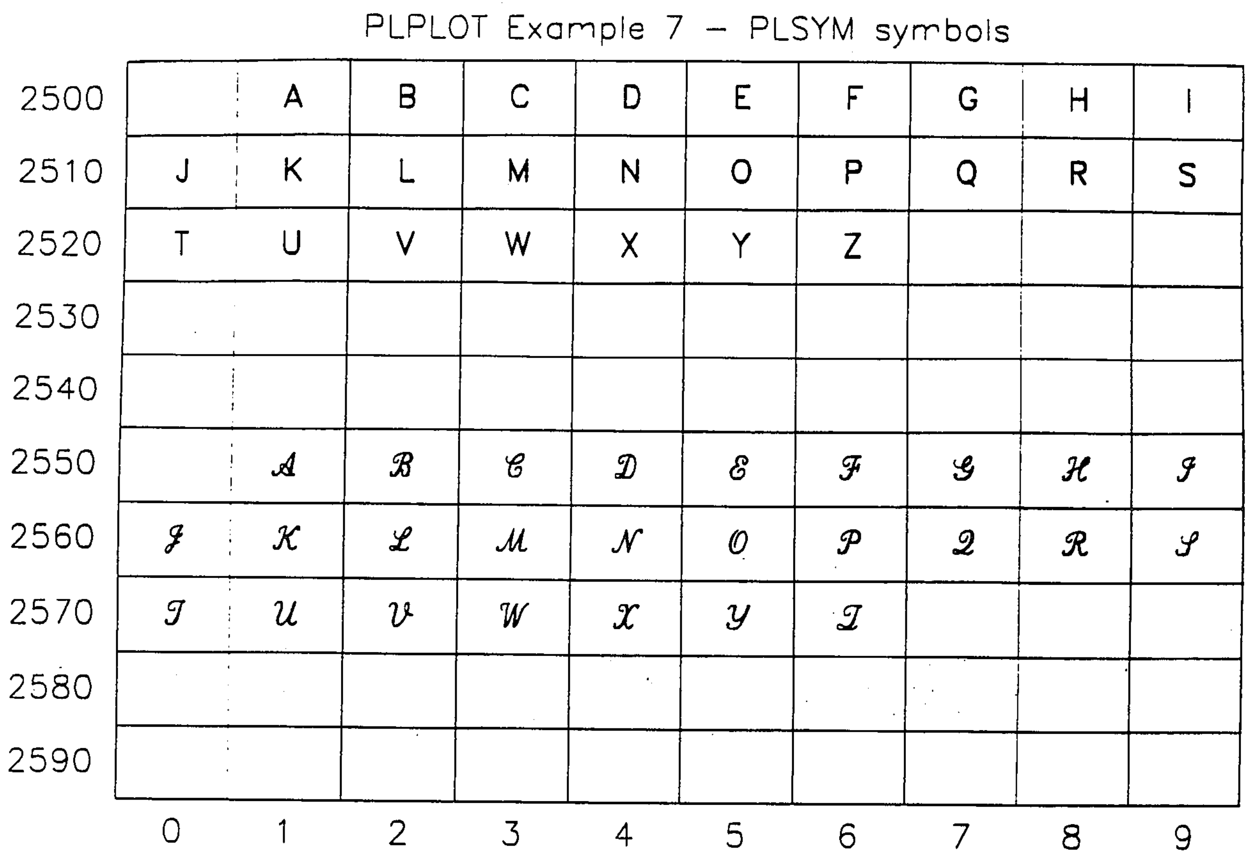
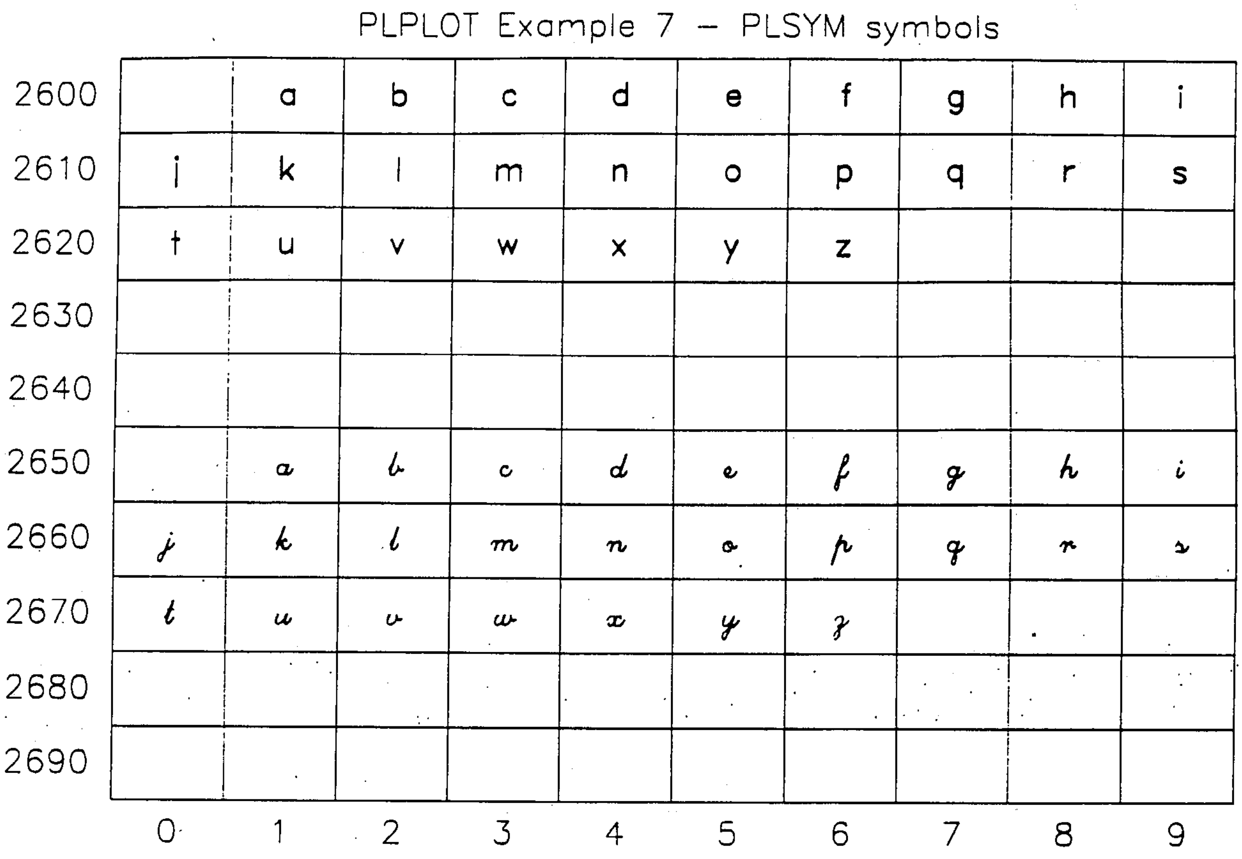

PLPLOT Example 7 - PLSYM symbols
2500
2510
2520
2530
2540
2550
2560
2570
2580
2590
A B C D E F G H I
J K L M N O P Q R S
T U V W X Y Z
𝒜 ℬ 𝒞 𝒟 ℰ ℱ 𝒢 ℋ ℐ
𝒥 𝒦 ℒ ℳ 𝒩 𝒪 𝒫 𝒬 ℛ 𝒮
𝒯 𝒰 𝒱 𝒲 𝒳 𝒴 𝒵
0 1 2 3 4 5 6 7 8 9

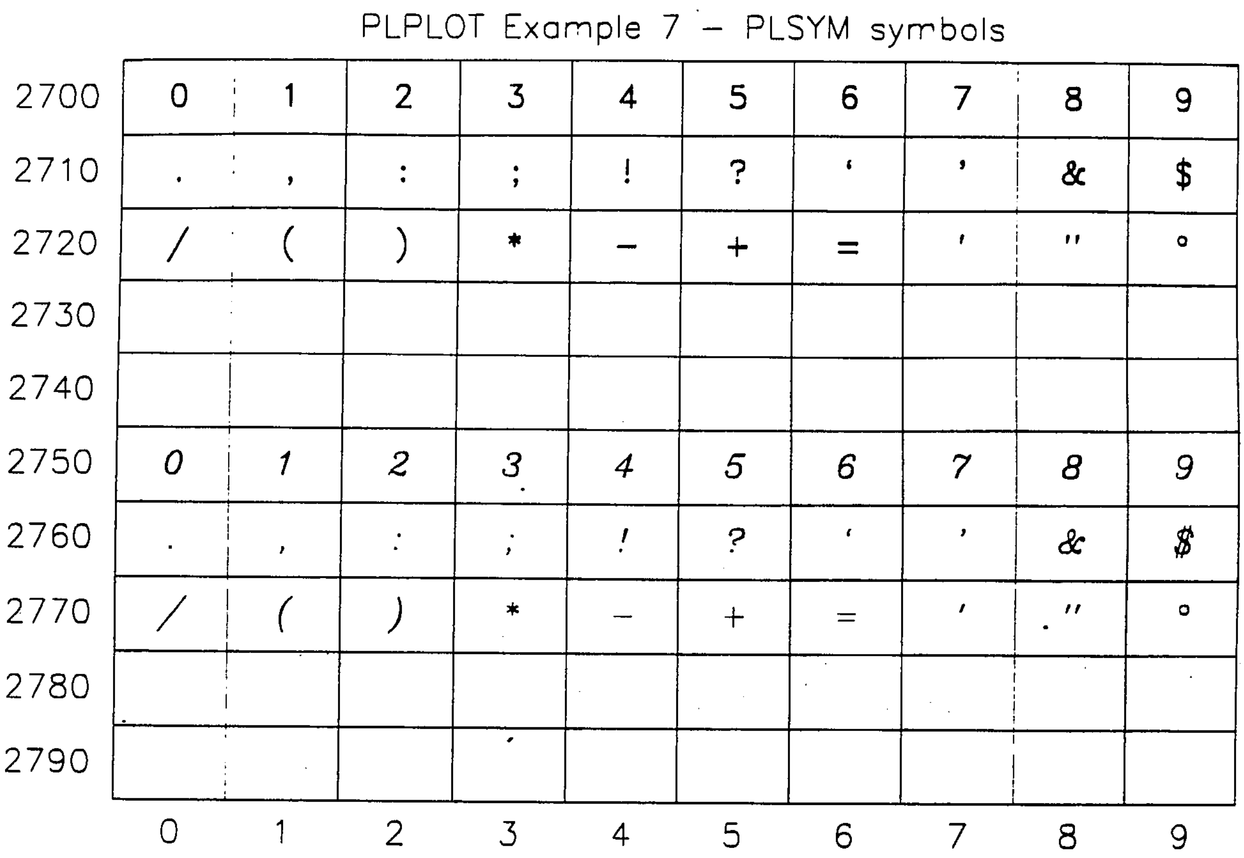
Fig. 7.13



PLPLOT Example 7 – PLSYM symbols

	0	1	2	3	4	5	6	7	8	9
2600		a	b	c	d	e	f	g	h	i
2610	j	k	l	m	n	o	p	q	r	s
2620	t	u	v	w	x	y	z			
2630										
2640										
2650		*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*
2660	*j*	*k*	*l*	*m*	*n*	*o*	*p*	*q*	*r*	*s*
2670	*t*	*u*	*v*	*w*	*x*	*y*	*z*			
2680										
2690										

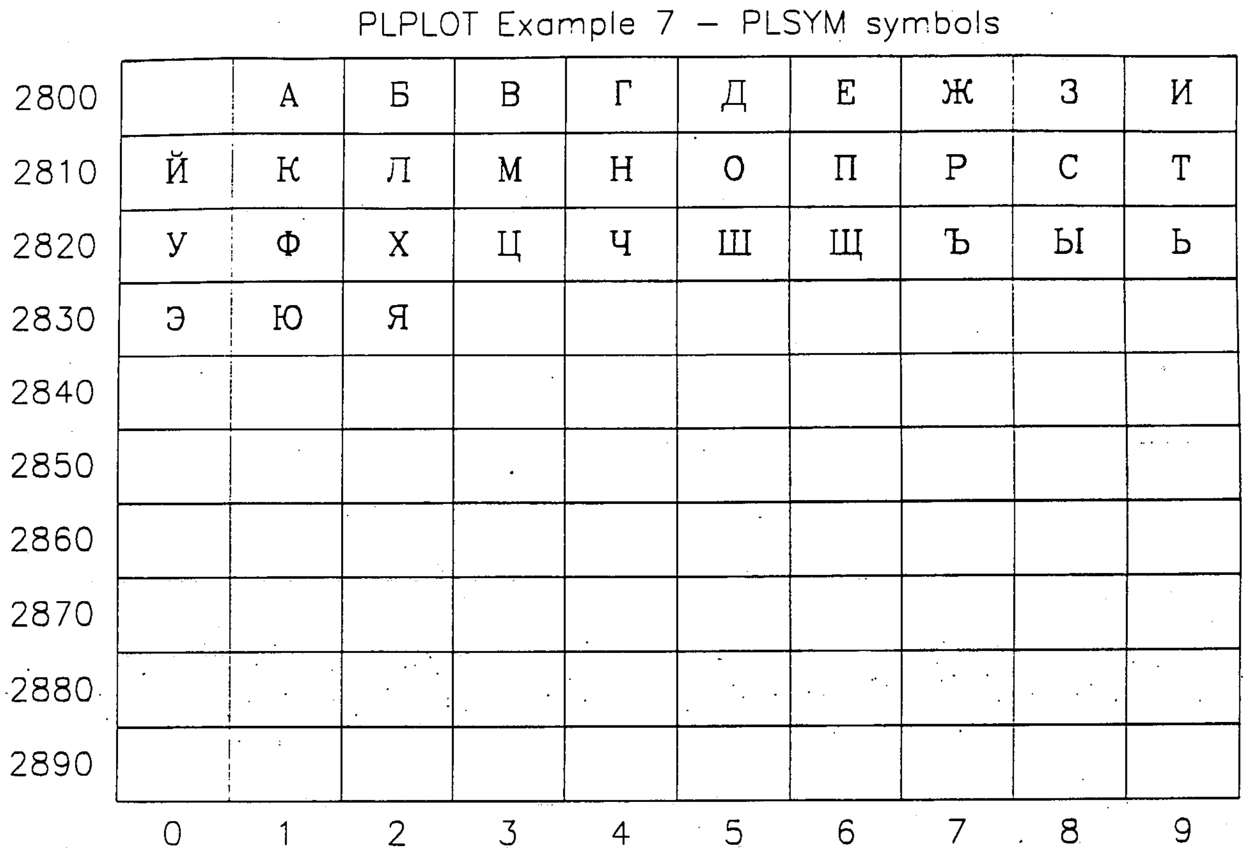
Fig. 7.14



PLPLOT Example 7 – PLSYM symbols

	0	1	2	3	4	5	6	7	8	9
2700	0	1	2	3	4	5	6	7	8	9
2710	.	,	:	;	!	?	‘	’	&	$
2720	/	(	)	*	−	+	=	′	″	°
2730										
2740										
2750	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
2760	*.*	*,*	*:*	*;*	*!*	*?*	*‘*	*’*	*&*	*$*
2770	*/*	*(*	*)*	*	−	+	=	′	″	°
2780										
2790										

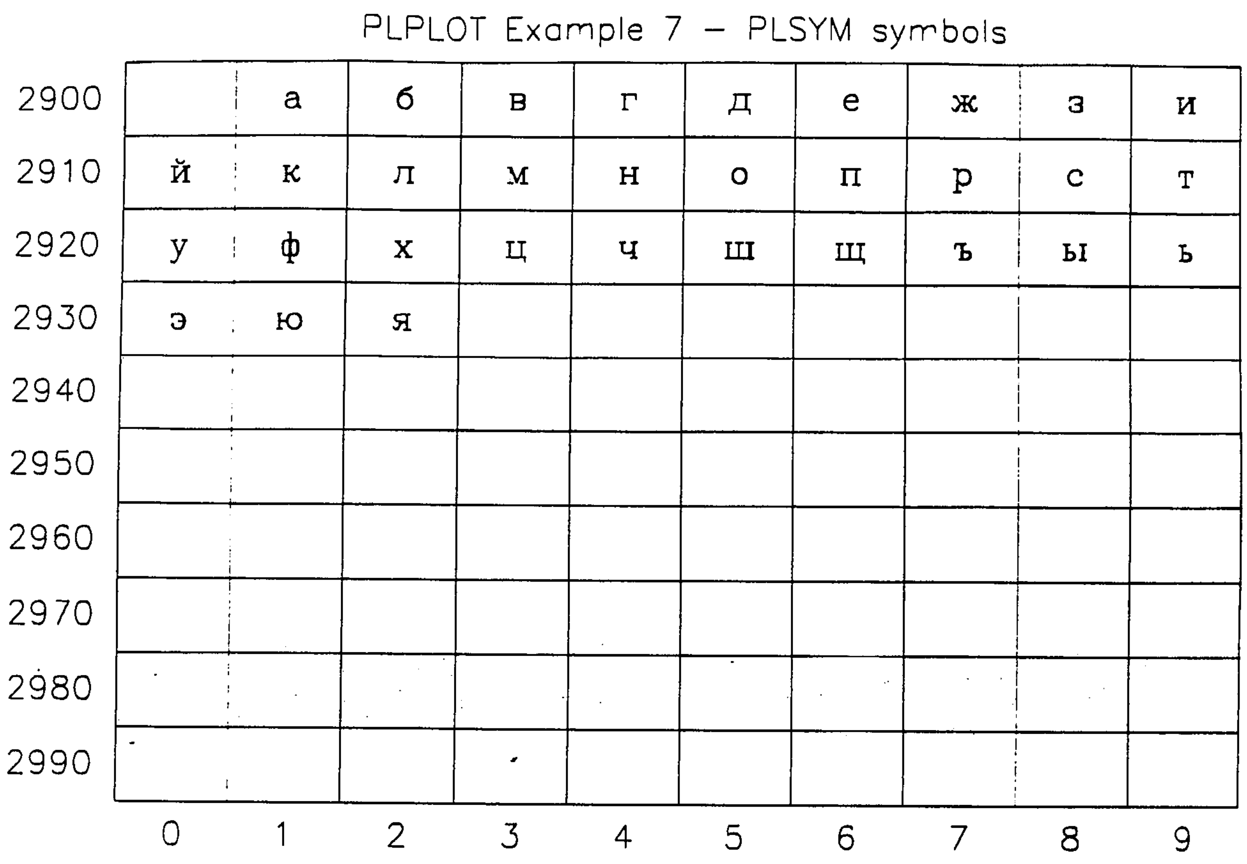
Fig. 7.15



PLPLOT Example 7 – PLSYM symbols

	0	1	2	3	4	5	6	7	8	9
2800		А	Б	В	Г	Д	Е	Ж	З	И
2810	Й	К	Л	М	Н	О	П	Р	С	Т
2820	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь
2830	Э	Ю	Я							
2840										
2850										
2860										
2870										
2880										
2890										

Fig. 7.16



PLPLOT Example 7 – PLSYM symbols

	0	1	2	3	4	5	6	7	8	9
2900		а	б	в	г	д	е	ж	з	и
2910	й	к	л	м	н	о	п	р	с	т
2920	у	ф	х	ц	ч	ш	щ	ъ	ы	ь
2930	э	ю	я							
2940										
2950										
2960										
2970										
2980										
2990										

Fig. 7.17

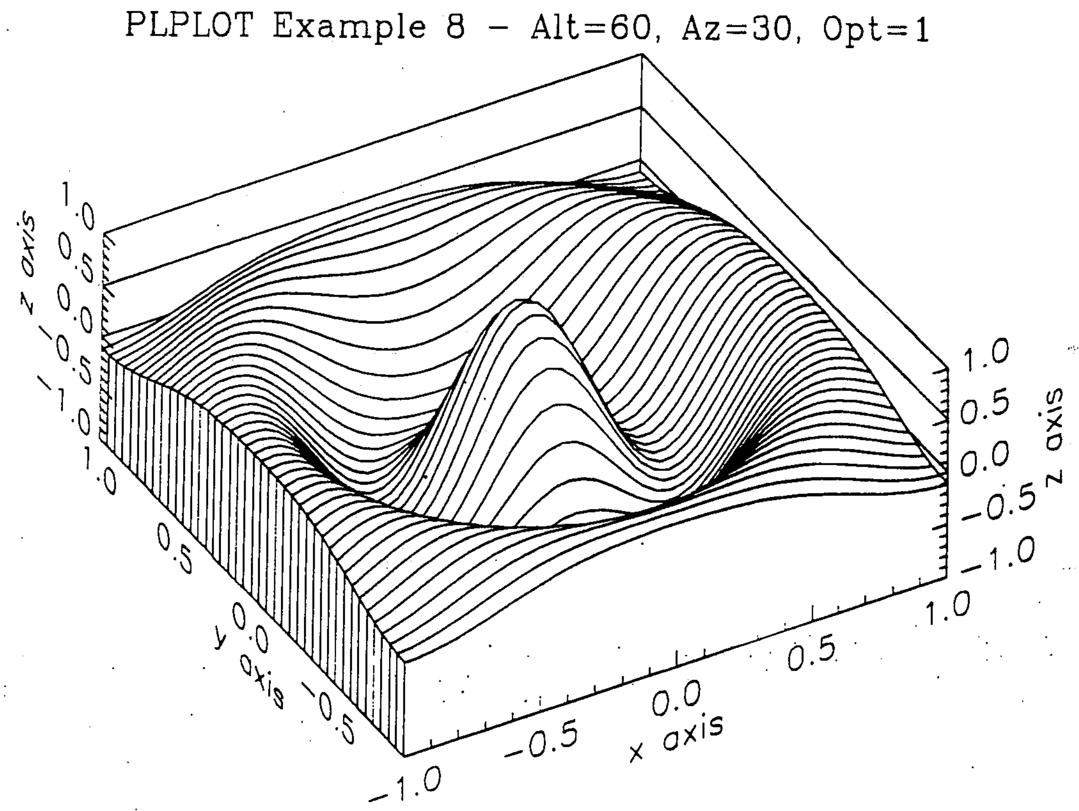

PLPLOT Example 8 - Alt=60, Az=30, Opt=1
z axis
1.0
0.5
0.0
-0.5
-1.0
1.0
0.5
0.0
-0.5
-1.0
z axis
1.0
0.5
0.0
-0.5
-1.0
y axis
-1.0
-0.5
0.0
0.5
1.0
x axis

Fig. 8.1

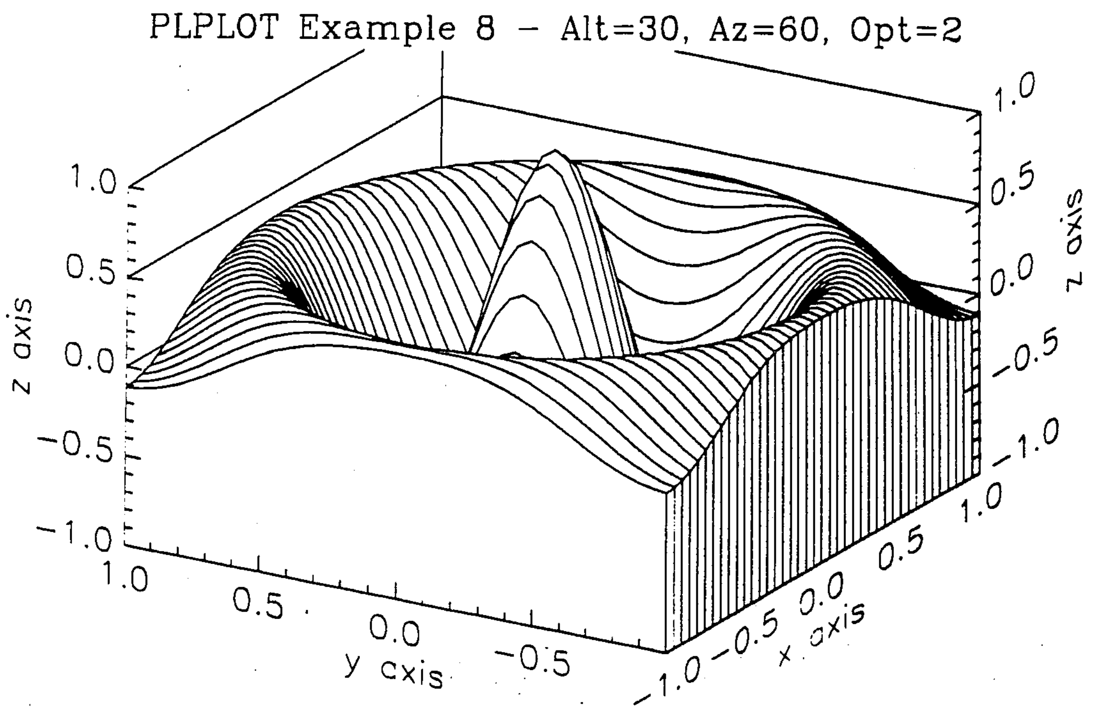

PLPLOT Example 8 - Alt=30, Az=60, Opt=2
1.0
0.5
0.0
-0.5
-1.0
z axis
1.0
0.5
0.0
-0.5
-1.0
z axis
1.0
0.5
0.0
-0.5
-1.0
y axis
-1.0
-0.5
0.0
0.5
1.0
x axis

Fig. 8.2

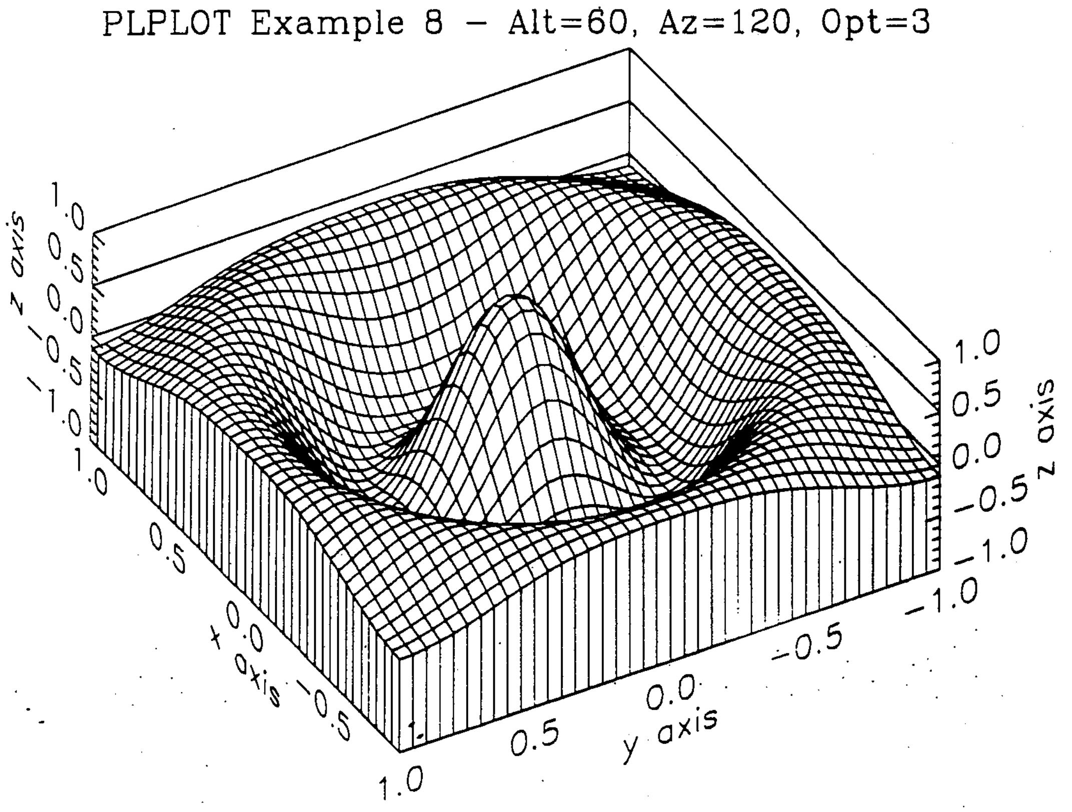

PLPLOT Example 8 - Alt=60, Az=120, Opt=3
z axis
1.0
0.5
0.0
-0.5
-1.0
x axis
1.0
0.5
0.0
-0.5
-1.0
y axis
1.0
0.5
0.0
-0.5
-1.0
z axis
1.0
0.5
0.0
-0.5
-1.0

Fig. 8.3

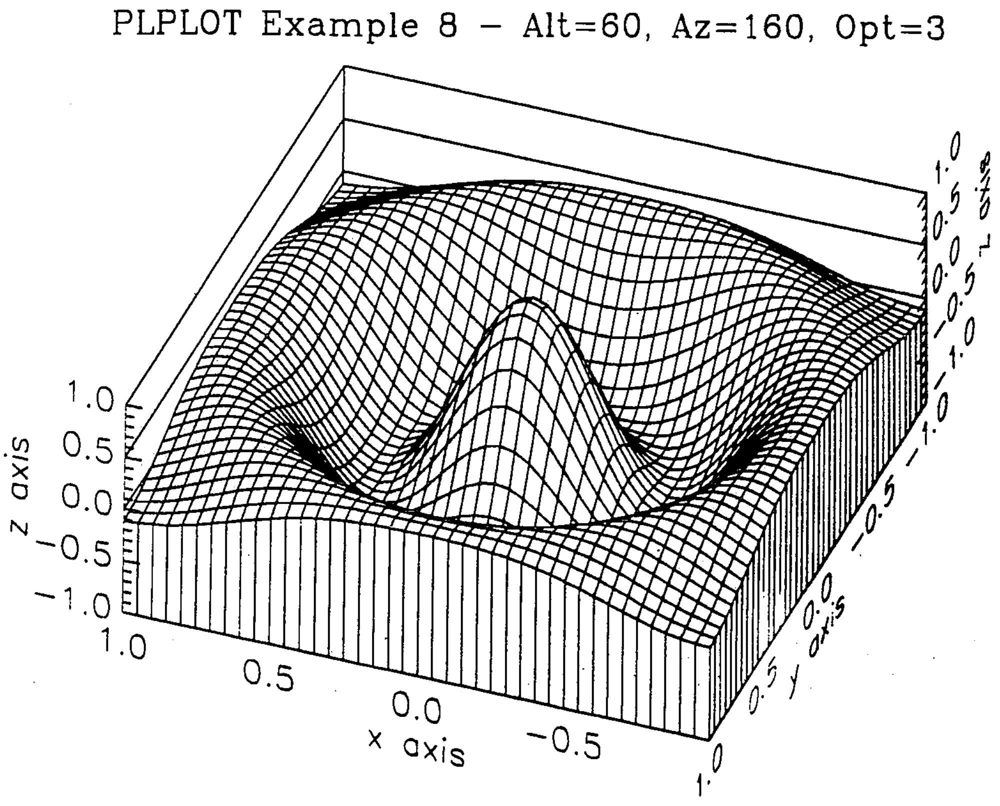
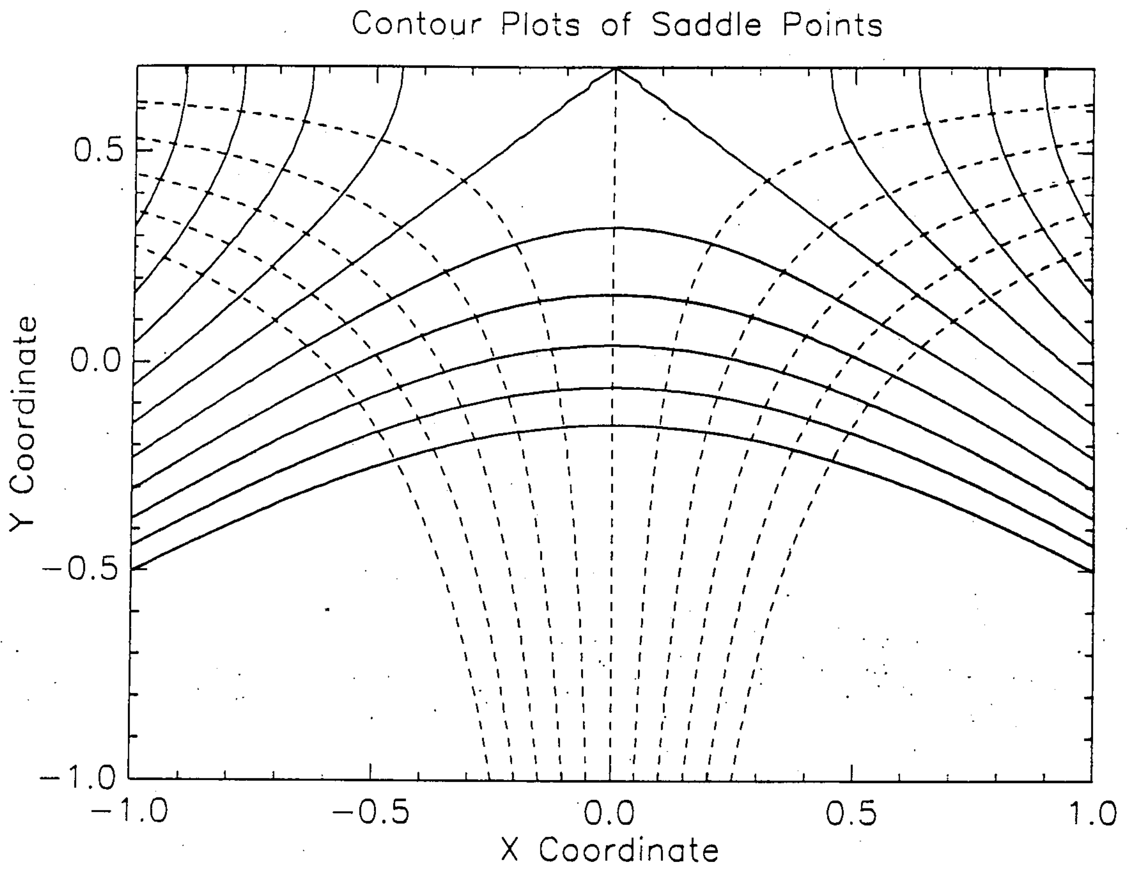

PLPLOT Example 8 - Alt=60, Az=160, Opt=3
z axis
1.0
0.5
0.0
−0.5
−1.0
x axis
1.0
0.5
0.0
−0.5
y axis
1.0
0.5
0.0
−0.5
−1.0
z axis
1.0
0.5
0.0
−0.5
−1.0

Fig. 8.4



Contour Plots of Saddle Points
Y Coordinate
X Coordinate
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

Fig. 9.1

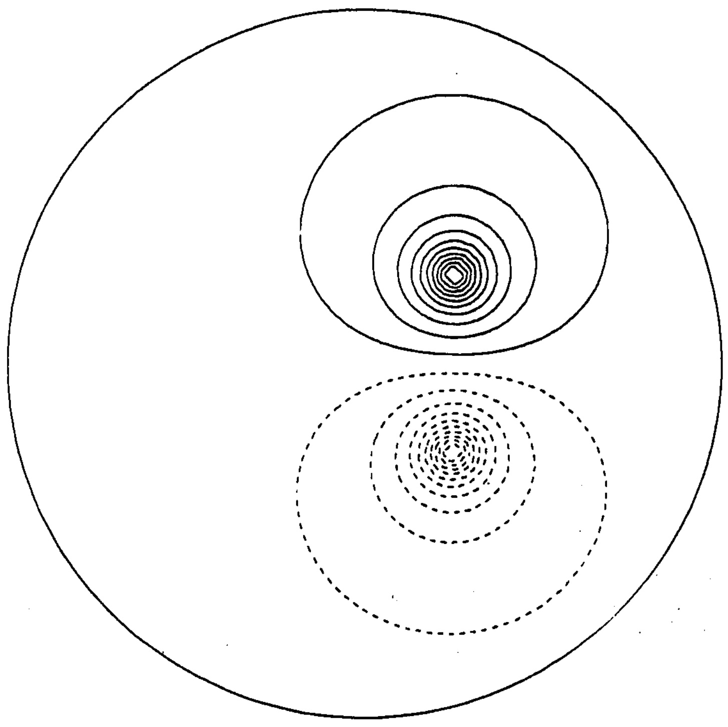


Fig. 9.2

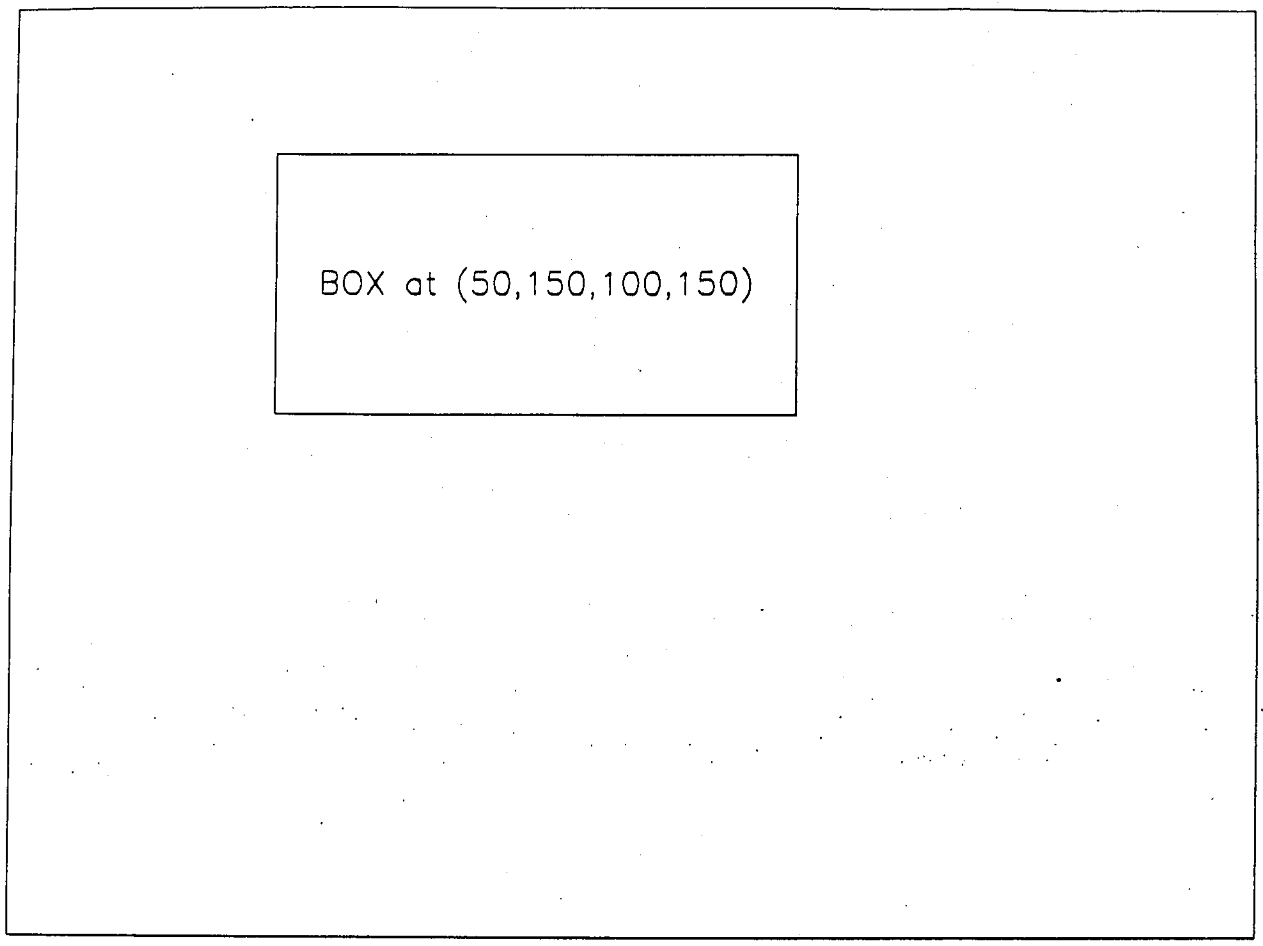


Fig. 10

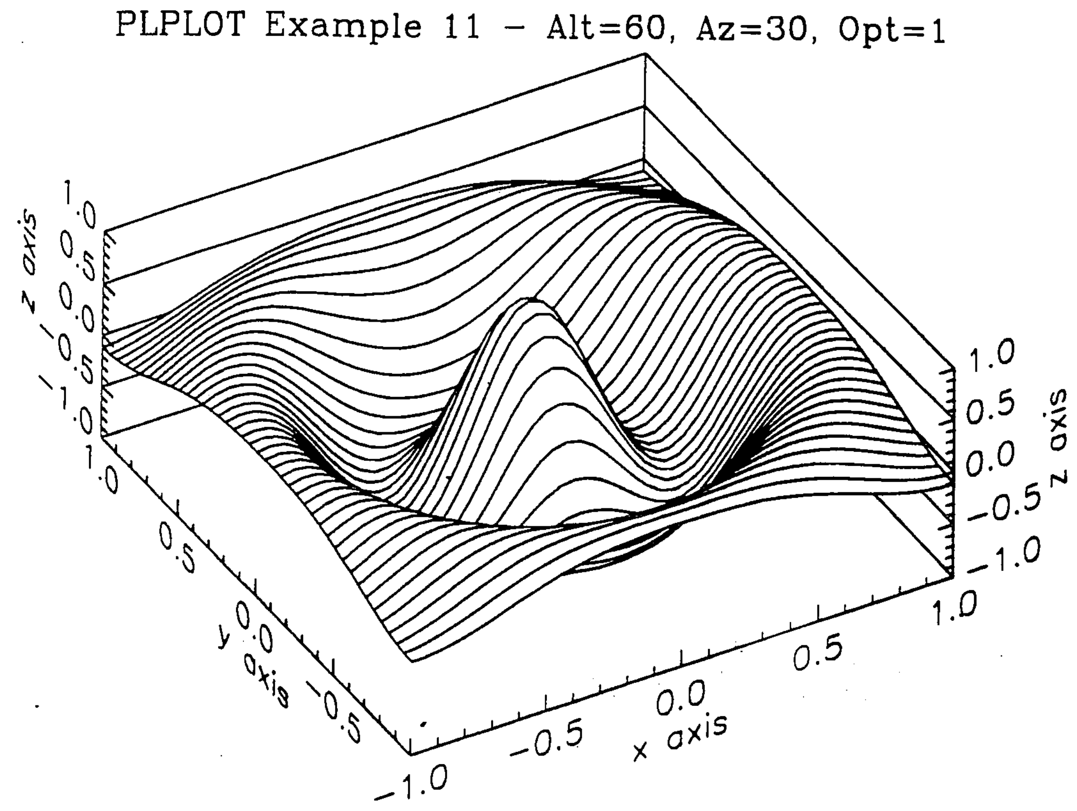

PLPLOT Example 11 - Alt=60, Az=30, Opt=1
z axis
1.0
0.5
0.0
-0.5
-1.0
y axis
1.0
0.5
0.0
-0.5
-1.0
x axis
-0.5
0.0
0.5
1.0
z axis
1.0
0.5
0.0
-0.5
-1.0

Fig. 11.1

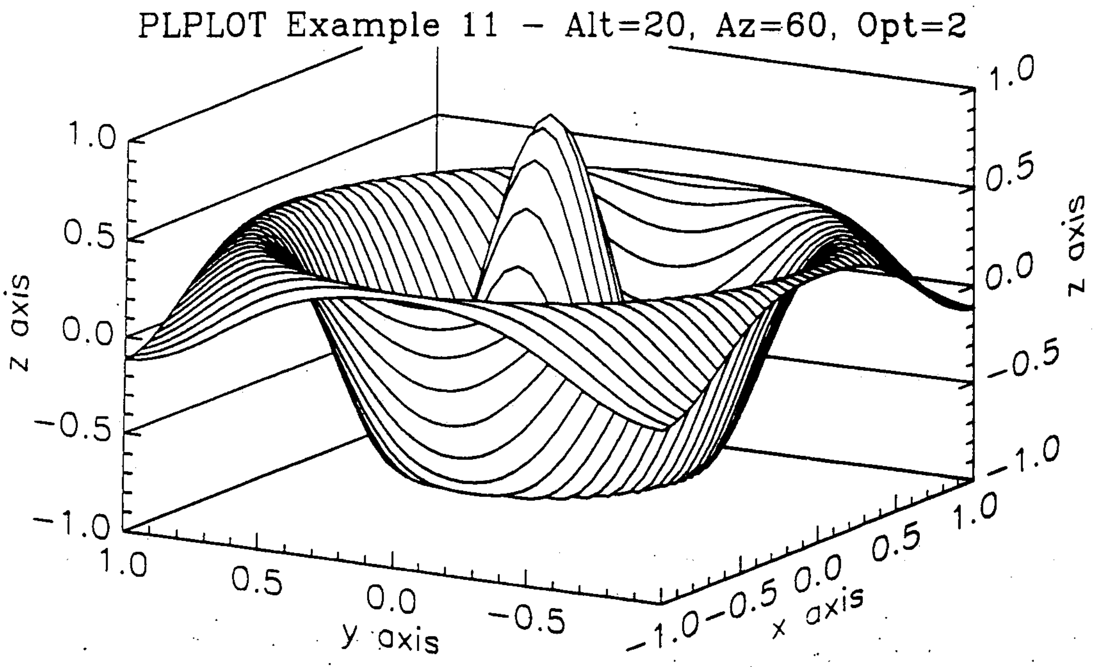
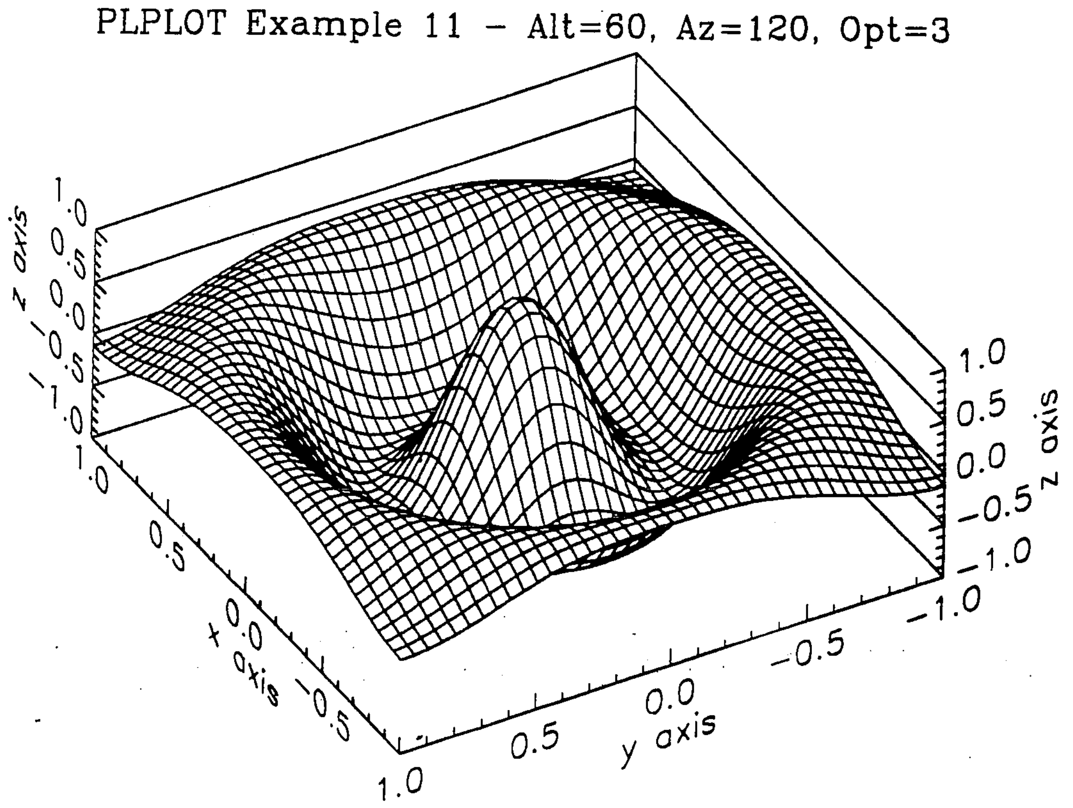

PLPLOT Example 11 - Alt=20, Az=60, Opt=2
z axis
1.0
0.5
0.0
−0.5
−1.0
y axis
x axis

Fig. 11.2



PLPLOT Example 11 - Alt=60, Az=120, Opt=3
z axis
1.0
0.5
0.0
−0.5
−1.0
x axis
1.0
0.5
0.0
−0.5
−1.0
y axis
1.0
0.5
0.0
−0.5
−1.0

Fig. 11.3

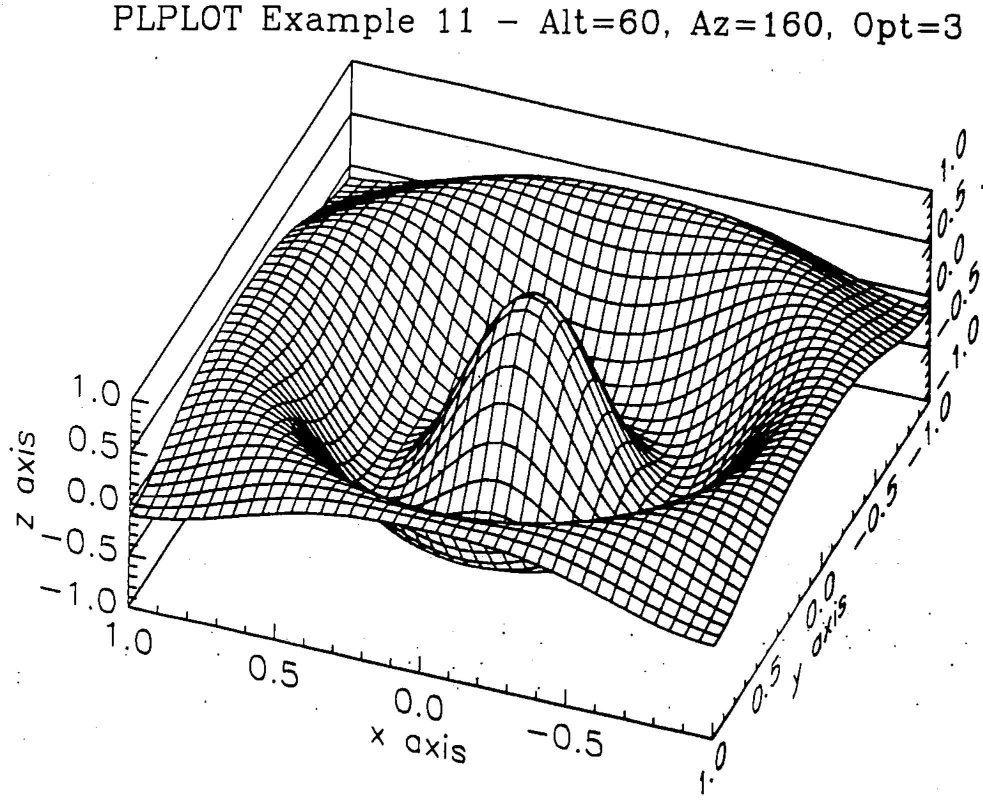

PLPLOT Example 11 - Alt=60, Az=160, Opt=3
1.0
0.5
0.0
-0.5
-1.0
z axis
1.0
0.5
0.0
-0.5
x axis
-1.0
-0.5
0.0
0.5
1.0
y axis

Fig. 11.4

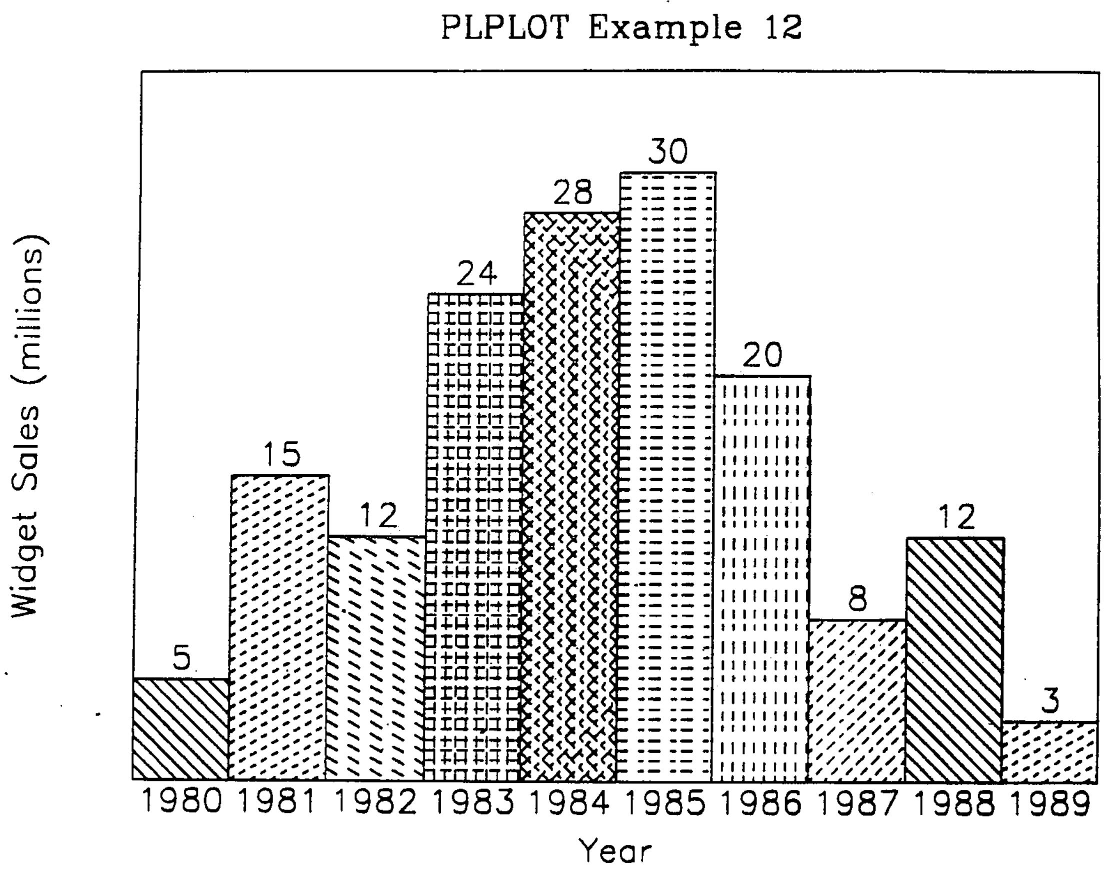

PLPLOT Example 12
Widget Sales (millions)
5
15
12
24
28
30
20
8
12
3
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
Year

Fig. 12

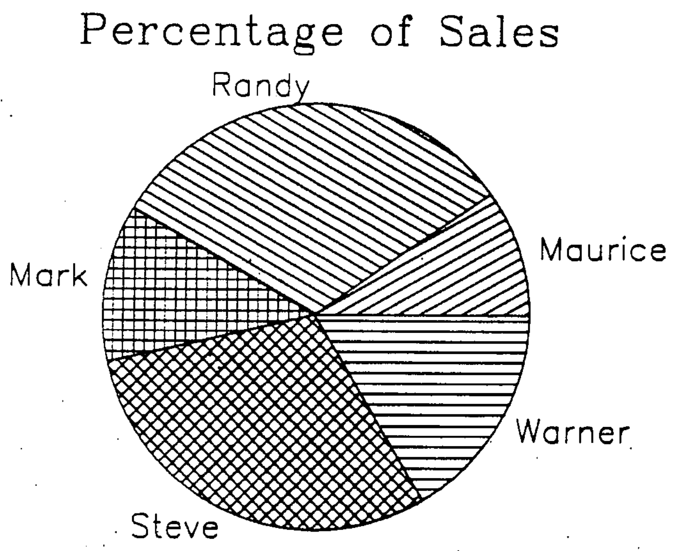


Fig. 13