PGPLOT

GRAPHICS SUBROUTINE LIBRARY

T. J. Pearson

June 1989

California Institute of Technology
Pasadena, California 91125
(818) 395–4980

PGPLOT, Version 4.9, June 1989
(Subroutine descriptions updated to Version 4.9D)

Please address comments on PGPLOT and this manual, including requests for copies, bug reports, and requests for improvements, to the author:

Tim Pearson,
Astronomy Department,
Caltech 105–24,
Pasadena 91125,
USA.

Telephone: +1 818 395-4980
INTERNET: tjp@astro.caltech.edu

Contents

Chapter 1

INTRODUCTION

1.1 PGPLOT

PGPLOT is a Fortran subroutine package for drawing simple scientific graphs on various graphics display devices. It was originally developed for use with astronomical data reduction programs in the Caltech Astronomy department.

This manual is intended for the Fortran programmer who wishes to write a program generating graphical output. For most applications, the program can be device-independent, and the output can be directed to the appropriate device at run time. The output device is described by a "device specification," discussed below. The programmer can build a specific device specification into the program, but it is better to make this a parameter which the user of the program can supply.

All the examples in this manual use standard Fortran-77. PGPLOT itself is written mostly in standard Fortran-77, with a few non-standard, system-dependent subroutines. At Caltech, it runs under the VAX/VMS, Convex-UNIX, and Sun-UNIX operating systems.

1.2 This Manual

This manual is intended both as a tutorial introduction to PGPLOT and as a reference manual. The remainder of this chapter describes some fundamentals: how to include the PGPLOT library in your program, and the types of graphic devices that PGPLOT can use.

Chapter 2 is tutorial: it presents a Fortran program for drawing a graph using the minimum number of PGPLOT subroutines, and explains what each of these subroutines does. After reading this chapter, you should be able to write your own PGPLOT program, although it may be helpful to refer to the individual subroutine descriptions in Appendix A.

The basic features of PGPLOT are introduced in Chapters 3, 4, and 5. Chapter 3 explains the positioning and scaling of plots on the page, Chapter 4 describes the basic ("primitive") routines for drawing lines, writing text, drawing graph markers, and shading areas, and Chapter 5 describes the routines for changing the "attributes" of these primitives: color, line-style, line-width, text font, etc.

Chapter 6 describes some "high level" routines that use the primitive routines to build up more complicated pictures: *e.g.*, function plots, histograms, bar charts, and contour maps.

Chapter 7 describes PGPLOT's capabilities for "interactive" graphics, whereby the user of the PGPLOT program can control its action with a cursor, joystick, mouse, etc.

Chapter 8 describes the use of "metafiles". A metafile is a disk file in which a device-independent representation of a graphics image can be stored. A translation program allows the image to be displayed on any supported device.

There are six appendices. Appendix A is a list of all the PGPLOT routines, with detailed instructions for their use. Appendix B shows the complete set of PGPLOT characters and symbols that can be used for annotating graphs. Appendix C is intended for those who want to install PGPLOT on another machine. Appendix D gives details of the devices supported by PGPLOT. Appendix E provides instructions for programmers who want to extend PGPLOT to support other devices. Appendix F describes how PGPLOT subroutines can be called from a program written in the C language.

1.3 Using PGPLOT

In order to use PGPLOT subroutines, you will need to link your program with the graphics subroutine library.

VAX/VMS On the Caltech Astronomy VAX computers, the graphics subroutine library is scanned automatically by the `LINK` command, so the following sequence of instructions suffices to compile, link, and run a graphics program `EXAMPLE.FOR`:

```
$ FORTRAN EXAMPLE
$ LINK EXAMPLE
$ RUN EXAMPLE
```

On other VMS computers, the automatic search of the graphics library may not occur. You will then need to include the graphics library explicitly by using a `LINK` commands like the following:

```
$ LINK EXAMPLE,PGPLOT_DIR:GRPSHR/LIB
```

The PGPLOT subroutines are not included in your `.EXE` file, but are fetched from a *shareable image* when you execute the `RUN` command. This makes the `.EXE` file much smaller, and means that the program need not be re-linked when changes are made to the graphics subroutines; but the `.EXE` file can only be run on a machine that has a copy of the shareable image and is running a compatible version of VAX/VMS. For more information, see Appendix C.

Unix The following assumes that the PGPLOT library `libpgplot.a` has been installed in a standard location where the loader can find it. To compile, link, and run a graphics program `example.f`:

```
fc -o example example.f -lpgplot
example
```

Unlike the VMS version, the PGPLOT routines are included in the executable file.

1.4 Graphics Devices

Graphics devices fall into two classes: devices which produce a hardcopy output, usually on paper; and interactive devices, which usually display the plot on a TV monitor. Some of the interactive devices allow modification to the displayed picture, and some have a movable cursor which can be used as a graphical input device. There is also a "null device," to which unwanted graphical output can be directed. Hardcopy devices are not used interactively. One must first create a disk file and then send it to the appropriate device with a print or copy command. Consult Appendix D (or your System Manager) to determine the appropriate device-specific command.

A PGPLOT graphical output device is described by a "device specification" that consists of two parts, separated by a slash (/): the *device name* or *file name*, and the *device type*.

Device name The device name or file name is the name by which the output device is known to the operating system. For most hardcopy devices, this should be the name of a disk file, while for interactive devices, it should be the name of a device of the appropriate type; in both cases, the name should be specified according to the syntax of the operating system in use. If the device or file name includes a slash (`/`), enclose the name in double quotation marks (`"`). If the device name is omitted from the device specification, a default device is used, the default depending on the device type (see Appendix D). In Unix, device and file names are case-sensitive.

Device type The device type tells PGPLOT what sort of graphical device it is. Appendix D lists the device types available at the time of writing, together with the names by which they are known to PGPLOT. If the device type is omitted, a system-dependent default type is assumed (this is the value of the "environment variable" `PGPLOT_TYPE`, and on Phobos and Deimos it is "Printronix"). The device type is not case-sensitive: you can use uppercase or lowercase letters, or a mixture of the two.

Examples (VMS)

Tektronix 4006/4010 terminal: `TTA4/TEK` (device `_TTA4:`).

Grinnell image display: `/GRIN`.

Disk file, Printronix format: `SYS$SCRATCH:PLOT.DAT/PRIN`.

Disk file, Versatec format, with the output file on a different DECnet node: `DEIMOS::XPLOT.DAT/PRIN`.

Disk file in default format in default directory: `PGPLOT.LIS`.

Examples (Unix)

Tektronix 4006/4010 terminal: `/TEK` (the logged-in terminal).

IVAS image display: `/IVAS`.

Disk file, Printronix format: `"/scr/tjp/plot.dat"/PRIN`.

Disk file in default format in default directory: `pgplot.lis`.

1.5 Environment variables

The run-time behavior of PGPLOT can be modified by defining one or more *environment variables*. The variables have names which begin with `PGPLOT_`. In VMS, they are logical names; in Unix, they are Unix environment variables.

To set the value of a variable in VMS (DCL):

```
$ DEFINE PGPLOT_ENVOPT VG
```

In Unix (csh):

```
setenv PGPLOT_ENVOPT VG
```

Quotation marks may be required around the value (double-quotes in VMS, single quotes in Unix) to prevent interpretation of special characters by the command interpreter.

To unset a variable in VMS or Unix:

```
$ DEASSIGN PGPLOT_ENVOPT
unsetenv PGPLOT_ENVOPT
```

The following are some of the environment variables currently in use:

- `PGPLOT_ENVOPT`: this variable provides additional options for the `PGENV` subroutine (see description in Appendix A).
- `PGPLOT_FONT`: the name of the binary file containing character font digitization, *e.g.*, `PGPLOT_FONT = "/usr/tjp/grfont.dat"`.
- `PGPLOT_IDENT`: if this variable is defined (with any value), the user name and time are written at the lower right corner of the plot by routine `PGEND` (hardcopy devices only), *e.g.*, `PGPLOT_IDENT = YES`.
- `PGPLOT_TYPE`: the plot type to be used in PGPLOT when a device specification omits the type, *e.g.*, `PGPLOT_TYPE = QMS`.
- `PGPLOT_BUFFER`: controls buffering (see Chapter 7).

Chapter 2

SIMPLE USE OF PGPLOT

2.1 Introduction

This chapter introduces the basic subroutines needed to create a graph using PGPLOT, by way of a concrete example. It does not describe all the capabilities of PGPLOT; these are presented in later chapters.

A graph is composed of several elements: a box or axes delineating the graph and indicating the scale, labels if required, and one or more points or lines. To draw a graph you need to call at least four of the PGPLOT subroutines:

1. `PGBEGIN`, to start up PGPLOT and specify the device you want to plot on;
2. `PGENV`, to define the range and scale of the graph, and draw labels, axes etc;
3. one or more calls to `PGPOINT` or `PGLINE` or both, or other drawing routines, to draw points or lines.
4. `PGEND` to close the plot.

To draw more than graph on the same device, repeat steps (2) and (3). It is only necessary to call `PGBEGIN` and `PGEND` once each, unless you want to plot on more than one device.

This chapter presents a very simple example program to demonstrate the above four steps.

2.2 An Example

A typical application of PGPLOT is to draw a set of measured data points and a theoretical curve for comparison. This chapter describes a simple program for drawing such a plot; in this case there are five data points and the theoretical curve is $y = x^2$. Here is the complete Fortran code for the program:

```
      PROGRAM SIMPLE
      REAL XR(100), YR(100)
      REAL XS(5), YS(5)
      DATA XS/1.,2.,3.,4.,5./
      DATA YS/1.,4.,9.,16.,25./
      CALL PGBEGIN(0,'?',1,1)
      CALL PGENV(0.,10.,0.,20.,0,1)
      CALL PGLABEL('(x)', '(y)', 'A Simple Graph')
      CALL PGPOINT(5,XS,YS,9)
      DO 10 I=1,60
          XR(I) = 0.1*I
          YR(I) = XR(I)**2
   10 CONTINUE
      CALL PGLINE(60,XR,YR)
      CALL PGEND
      END
```

The following sections of this chapter describe how the program works, and the resulting plot is shown in Figure 2.1.

2.3 Data Initialization

We shall store the x and y coordinates of the five data points in arrays `XS` and `YS`. For convenience, this program defines the values in `DATA` statements, but a more realistic program might read them from a file. Arrays `XR` and `YR` will be used later in the program for the theoretical curve.

```
      REAL XR(100), YR(100)
      REAL XS(5), YS(5)
      DATA XS/1.,2.,3.,4.,5./
      DATA YS/1.,4.,9.,16.,25./
```

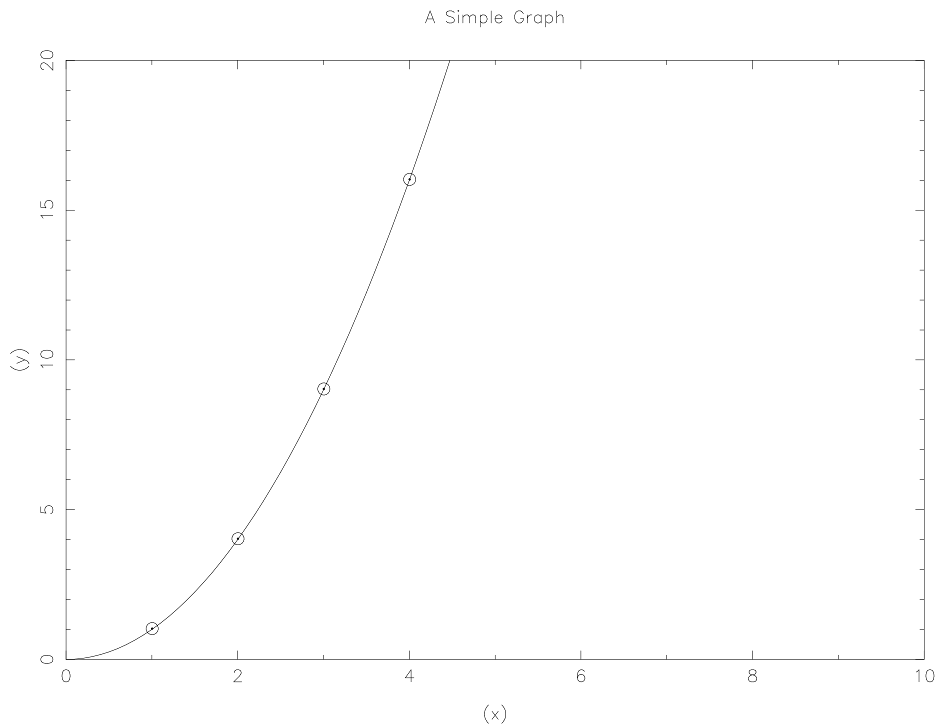

Figure 2.1 Output from example program.

2.4 Starting PGPLOT

The first thing the program must do is to start up PGPLOT and select the graphics device for output:

```
CALL PGBEGIN(0,'?',1,1)
```

Subroutine `PGBEGIN` has four arguments:

1. The first argument is present for historical reasons. It should always be set to zero (0).
2. The second argument is a character string which gives a "device specification" for the interactive graphics device or disk file for hardcopy graphics (see Chapter 1 and Appendix D). This program makes use of a special shorthand feature of PGPLOT, however: if this argument is set to `'?'`, the program will ask the user to supply the device specification at run-time.

3, 4. The last two arguments are described in §3.2. Usually they are both set to 1, as in this example.

2.5 Defining Plot Scales and Drawing Axes

Subroutine PGENV starts a new picture and defines the range of variables and the scale of the plot. PGENV also draws and labels the enclosing box and the axes if requested. In this case, the x-axis of the plot will run from 0.0 to 10.0 and the y axis will run from 0.0 to 20.0.

```
CALL PGENV(0.,10.,0.,20.,0,1)
```

PGENV has six arguments:

1, 2. the left and right limits for the x (horizontal) axis (real numbers, not integers).

3, 4. the bottom and top limits for the y (vertical) axis (also real numbers).

5. If this (integer) argument is 1, the scales of the x-axis and y-axis (in units per inch) will be equal; otherwise the axes will be scaled independently. In this case we have not requested equal scales.
6. This argument controls whether an enclosing box, tick-marks, numeric labels, and/or a grid will be put on the graph. The recommended value is 0. Some of the allowed values are:

   -2: no annotation;

   -1: draw box only;

   0: draw box, and label it with coordinate values around the edge;

   1: in addition to the box and labels, draw the two axes (lines $x = 0$, $y = 0$) with tick marks;

   2: in addition to the box, labels, and axes, draw a grid at major increments of the x and y coordinates.

2.6 Labeling the Axes

Subroutine PGLABEL may (optionally) be called after PGENV to write identifying labels on the x and y axes, and at the top of the picture:

```
CALL PGLABEL('(x)', '(y)', 'A Simple Graph')
```

All three arguments are character variables or constants; any of them can be blank (' ').

1. A label for the x-axis (bottom of picture).
2. A label for the y-axis (left-hand edge).
3. A label for the plot (top of picture).

2.7 Drawing Graph Markers

Subroutine PGPOINT draws *graph markers* at one or more points on the graph. Here we use it to mark the five data points:

```
CALL PGPOINT(5,XS,YS,9)
```

If any of the specified points fall outside the window defined in the call to PGENV, they will not be plotted. The arguments to PGPOINT are:

1. The number of points to be marked (integer).
2, 3. The x and y coordinates of the points (real arrays).
4. The number of the symbol to be used to mark the points. In this example, we use symbol number 9 which is a circle with a central dot. The available symbols are shown in Chapter 4.

2.8 Drawing Lines

The following code draws the "theoretical curve" through the data points:

```
      DO 10 I=1,60
          XR(I) = 0.1*I
          YR(I) = XR(I)**2
   10 CONTINUE
      CALL PGLINE(60,XR,YR)
```

We compute the x and y coordinates at 60 points on the theoretical curve, and use subroutine PGLINE to draw a curve through them. PGLINE joins up the points with straight-line segments, so it is necessary to compute coordinates at fairly close intervals in order to get a smooth curve. Any lines which cross the boundary of the window defined in PGENV are "clipped" at the boundary, and lines which lie outside the boundary are not drawn. The arguments of PGLINE are like those of PGPOINT:

1. The number of points defining the line (integer).
2, 3. The x and y coordinates of the points (real arrays).

2.9 Ending the Plot

Subroutine PGEND must be called to complete the graph properly, otherwise some pending output may not get sent to the device:

```
CALL PGEND
```

2.10 Compiling and running the program

To compile the program and link it with the PGPLOT library, see Chapter 1. For example, under VMS:

```
$ EDIT SIMPLE.FOR
...
$ FORTRAN SIMPLE
$ LINK SIMPLE
```

Under Unix:

```
ed simple.f
...
fc -o simple simple.f -lpgplot
```

When you run the program, it will ask you to supply the graphics device specification. Type in any allowed device specification, or type a question-mark (`?`) to get a list of the available device types. For example, if you are using a VT125 terminal, type `/VT`: the graph will appear on the terminal screen.

If you want a hard copy, you can run the program again, and specify a different device type, e.g., `SIMPLE.PLT/VERS` to make a disk file in Versatec format. To obtain the hard copy, print the file (but first check with your system manager what the correct print command is; it is possible to waste a lot of paper by using the wrong command or sending a file to the wrong sort of printer!).

Chapter 3

WINDOWS AND VIEWPORTS

3.1 Introduction

This chapter is concerned with positioning a graph on the screen or hardcopy page, and controlling its scale. In simple applications, the position and scale of the graph are controlled more-or-less automatically by the routine `PGENV`, but in order to obtain complete control of positioning and scaling, it is necessary to understand the concepts of the *View Surface*, the *Window*, and the *Viewport*, and two coordinate systems: *World Coordinates* and *Device Coordinates*.

A simple PGPLOT picture might be a two-dimensional graph showing the dependence of one variable on another. A typical graph has data points, represented by error bars or special markers such as dots or diamonds, possibly connected by lines, or perhaps plotted on the same scale as a theoretical model drawn as a smooth curve. The graph must be labeled with two axes to indicate the coordinate scales.

The programmer must describe to PGPLOT the various elements of the graph in terms of rectangular Cartesian coordinates. The only limitation on the coordinates is that they be represented as floating-point (`REAL*4`) numbers; otherwise we are totally free to choose the meaning of the coordinates. For example, in a graph showing the temporal variation of a radio source, the abscissa (x-coordinate) might be Epoch (in years) and the ordinate (y-coordinate) Flux Density (in Jy).

In accordance with common practice in graphics programming, these coordinates, chosen by the programmer, are termed *world coordinates*. PGPLOT maps a selected rectangular region of the world-coordinate space (termed the *window*) onto a specified rectangle (termed the *viewport*) on the *view surface* (the screen of an interactive display or a sheet of paper on a hardcopy plotter). The program must make calls to PGPLOT routines to define both the window and the viewport. For complete descriptions of the routines and their arguments, refer to Appendix A.

3.2 Selecting a View Surface

The first thing a graphics program must do is to tell PGPLOT what device it is going to use. This is done by calling routine `PGBEGIN`. For example, to create a plot file for the Versatec printer:

```
CALL PGBEGIN (0, 'PLOTFILE.LIS/VERSATEC', 1, 1)
```

Equally important, when all plotting has been completed, it is necessary to call `PGEND` to flush any pending plot requests:

```
CALL PGEND
```

Note that only one device can be used at a time. If `PGBEGIN` is called while a plot is in progress, the old plot is closed and a new one is begun.

After calling `PGBEGIN` the program has access to a *view surface*. For interactive devices, this is the full screen of the device. For hardcopy devices, it is a standard page, usually $10(x) \times 8.5(y)$ inches on a device used in "landscape" mode (*e.g.*, device types `/VE` and `/QMS`), or $8.5(x) \times 10(y)$ inches a device used in "portrait" mode (*e.g.*, device types `/VV` and `/VQMS`).

On some devices, it is possible to plot on a larger piece of paper than the standard page; see the description of routine `PGPAPER`, which must be called immediately after `PGBEGIN` to change the size of the view surface. The different devices differ not only in the size of the view surface, but also in its *aspect ratio* (height/width). `PGPAPER` can be called to ensure that a plot has the same aspect ratio no matter what device it is plotted on.

After completing a graph, it is possible to advance to a new page to start a new graph (without closing the plot file) by calling `PGPAGE`:

```
CALL PGPAGE
```

This clears the screen on interactive devices, or gives a new piece of paper on hardcopy devices. It does not change the viewport or window.

The last two arguments of `PGBEGIN` (`NX` and `NY`) can be used to subdivide the view surface into smaller pieces called sub-pages, each of which can then be used separately. The view-surface is divided into NX (horizontally) by NY (vertically) sub-pages. When the view surface has been subdivided in this way, `PGPAGE` moves the plotter to the next sub-page, and only clears the screen or loads a new piece of paper if there are no sub-pages left on the current page.

In addition to selecting the view surface, `PGBEGIN` also defines a default viewport and window. It is good practice, however, to define the viewport and window explicitly as described below.

3.3 Defining the Viewport

A *viewport* is a rectangular portion of the plotting surface onto which the graph is mapped. PGPLOT has a default viewport which is centered on the plotting surface and leaves sufficient space around it for annotation. The application program can redefine the viewport by calling routine `PGVPORT` or `PGVSIZE`.

`PGVPORT` defines the viewport in a device-independent manner, using a coordinate system whose coordinates run from 0 to 1 in both x and y. This coordinate system is called *normalized device coordinate space*. For example, if we wish to divide the view surface into four quadrants and map a different plot onto each quadrant, we can define a new viewport before starting each plot. `PGVPORT` has the format:

```
CALL PGVPORT (XMIN, XMAX, YMIN, YMAX)
```

For example, to map the viewport onto the upper left quadrant of the view surface:

```
CALL PGVPORT (0.0, 0.5, 0.5, 1.0)
```

(Note that this does not leave room around the edge of the viewport for annotation.)

`PGVSIZE` defines the viewport in absolute coordinates (inches); it should only be used when it is known how big the view surface is and a definite plot scale is required. The arguments are the same as for `PGVPORT`, but measured in inches from the bottom left corner of the view surface. For example:

```
CALL PGVSIZE (1.5, 9.5, 1.5, 6.5)
```

defines a rectangular viewport 8 by 5 inches, offset 1.5 inches from the bottom and left edges of the view surface.

`PGVSTAND` defines a standard viewport, the size of which depends on the particular device being used, and on the current character size (it uses the whole view surface excluding a margin of four character heights all around):

```
CALL PGVSTAND
```

This is the default viewport set up by `PGBEGIN`.

Note that the viewport must be defined *before* calling any routines that would actually generate a display. The viewport may, however, be changed at any time: this will affect the appearance of objects drawn later in the program.

3.4 Defining the Window

The program defines the *window* by calling routine PGWINDOW, whose arguments specify the world-coordinate limits of the window along each coordinate axis. *e.g.*:

```
CALL PGWINDOW (1975.0, 1984.0, 5.0, 20.0)
```

specifies that the x-axis (epoch) is going to run (left to right) from 1975 to 1984, and the y-axis (flux density) is going to run (bottom to top) from 5 to 20 Jy. Note that the arguments are floating-point numbers (Fortran REAL variables or constants), and require decimal points. If the order of either the x pair or the y pair is reversed, the corresponding axis will point in the opposite sense, *i.e.*, right to left for x or top to bottom for y. PGPLOT uses the window specification to construct a mapping that causes the image of the window to coincide with the viewport on the view surface. Furthermore, PGPLOT "clips" lines so that only those portions of objects that lie within the window are displayed on the view surface.

Like the viewport, the window must be defined before drawing any objects. The window can be defined either before or after the viewport: the effect will be the same. The default window, set up by PGBEGIN, has x limits 0.0–1.0 and y limits 0.0–1.0.

If the ratio of the sides of the window does not equal the ratio of the sides of the viewport, the mapping of the world coordinates onto the view surface results in an image whose shape is compressed in either x or y. One way to avoid this compression is to carefully choose the viewport to have the same aspect ratio as the window. Routine PGWNAD can do this: it defines the window and simultaneously adjusts the viewport to have the same aspect ratio as the window. The new viewport is the largest that can fit inside the old one, and is centered in the old one.

3.5 Annotating the Viewport

For a simple graph, it is usually necessary to draw a frame around the viewport and label the frame with tick marks and numeric labels. This can be done with the routine PGBOX. For our sample graph, the call might be:

```
CALL PGBOX ('BCTN', 0.0, 0, 'BCNST', 0.0, 0)
```

Another routine, PGLABEL, provides text labels for the bottom, left hand side, and top of the viewport:

```
CALL PGLABEL ('Epoch', 'Flux Density (Jy)',
              'Variation of 3C345 at 10.7 GHz')
```

The first two arguments provide explanations for the two axes; the third provides a title for the whole plot. Note that unlike all the other plotting routines, the lines and characters drawn by PGBOX and PGLABEL are not clipped at the boundaries of the window. PGLABEL actually calls a more general routine, PGMTEXT, which can be used for plotting labels at any point relative to the viewport.

The amount of space needed outside the viewport for annotation depends on the exact options specified in PGBOX; usually four character heights will be sufficient, and this is the amount allowed when the standard viewport (created by PGVSTAND) is used. The character height can be changed by using routine PGSCH.

3.6 Routine PGENV

Having to specify calls to PGPAGE, PGVPORT, PGWINDOW, and PGBOX is excessively cumbersome for drawing simple graphs. Routine PGENV (for PGplot ENVironment) combines all four of these in one subroutine, using the standard viewport, and a limited set of the capabilities of PGBOX. For example, the graph described above could be initiated by the following call:

```
CALL PGENV (1975.0, 1984.0, 5.0, 20.0, 0, 0)
```

which is equivalent to the following series of calls:

```
CALL PGPAGE
CALL PGVSTAND
CALL PGWINDOW (1975.0, 1984.0, 5.0, 20.0)
CALL PGBOX ('BCNST', 0.0, 0, 'BCNST', 0.0, 0)
```

PGENV uses the standard viewport. The first four arguments define the world-coordinate limits of the window. The fifth argument can be 0 or 1; it is 1, PGENV calls PGWNAD instead of PGWINDOW so that the plot has equal scales in x and y. The sixth argument controls the amount of annotation.

Chapter 4

PRIMITIVES

4.1 Introduction

Having selected a view surface and defined the viewport and the window, we are ready to draw the substance of the image that is to appear within the viewport. This chapter describes the most basic routines, called *primitives*, that can be used for drawing elements of the image. There are four different sorts of primitive: *lines, graph-markers, text*, and *area fill*. Chapter 5 explains how to change the *attributes* of these primitives, *e.g.*, color, line-style, text font; and Chapter 6 describes some higher-level routines that simplify the composition of images that would require a large number of calls to the primitive routines.

The primitive routines can be used in any combination and order after the viewport and window have been defined. They all indicate where the primitive is to appear on the view surface by specifying world coordinates. See the subroutine descriptions in Appendix A for more details.

4.2 Clipping

The primitives are "clipped" at the edge of the viewport: any parts of the image that would appear outside the viewport are suppressed. The various primitives behave slightly differently. A *line* is clipped where it crosses the edge of the viewport. A *graph marker* is plotted if the center (the point marked) lies within or on the edge of the viewport; otherwise it is suppressed. *Text*, which is usually used for annotation, is not clipped (except at the edge of the view surface. A *filled area* is clipped at the edge of the viewport.

4.3 Lines

The primitive line-drawing routine is PGLINE. This draws one or more connected straight-line segments (generally called a *polyline* in computer graphics). It has three arguments: the number (N) of points defining the polyline, and two arrays (XPTS and YPTS) containing the world x and y-coordinates of the points. The polyline consists of $N-1$ straight-line segments connecting points 1–2, 2–3, ..., $(N-1)$–N:

```
CALL PGLINE (N, XPTS, YPTS)
```

The two routines PGMOVE and PGDRAW are even more primitive than PGLINE, in the sense that any line graph can be produced by calling these two routines alone. In general, PGLINE should be preferred, as it is more modular. PGMOVE and PGDRAW are provided for those who are used to Calcomp-style plotting packages. PGMOVE moves the plotter "pen" to a specified point, without drawing a line ("pen up"). It has two arguments: the world-coordinates of the required new pen position. PGDRAW moves the plotter "pen" from its current position (defined by the last call of PGMOVE or PGDRAW) to a new point, drawing a straight line as it goes ("pen down"). The above call to PGLINE could be replaced by the following:

```
CALL PGMOVE (XPTS(1), YPTS(1))
DO I=2,N
    CALL PGDRAW (XPTS(I), YPTS(I))
END DO
```

4.4 Graph Markers

A Graph Marker is a symbol, such as a cross, dot, or circle, drawn on a graph to mark a specific point. Usually the symbol used will be chosen to be symmetrical with a well-defined center. The routine PGPOINT draws one or more graph markers (sometimes called a *polymarker*). It has four arguments: the number (N) of points to mark, two arrays (XPTS and YPTS) containing the world x and y-coordinates of the points, and a number (NSYM) identifying the symbol to use:

```
CALL PGPOINT (N, XPTS, YPTS, NSYM)
```

The symbol number can be: -1, to draw a dot of the smallest possible size (one pixel); 0–31, to draw any one of the symbols in Figure 4.1; or 33–127, to draw the corresponding ASCII character (the character is taken from the currently selected text font); or > 127, to draw one of the Hershey symbols from Appendix B. The Fortran ICHAR function can be used to obtain the ASCII value; *e.g.*, to use letter F:

```
CALL PGPOINT (1, 0.5, 0.75, ICHAR('F') )
```

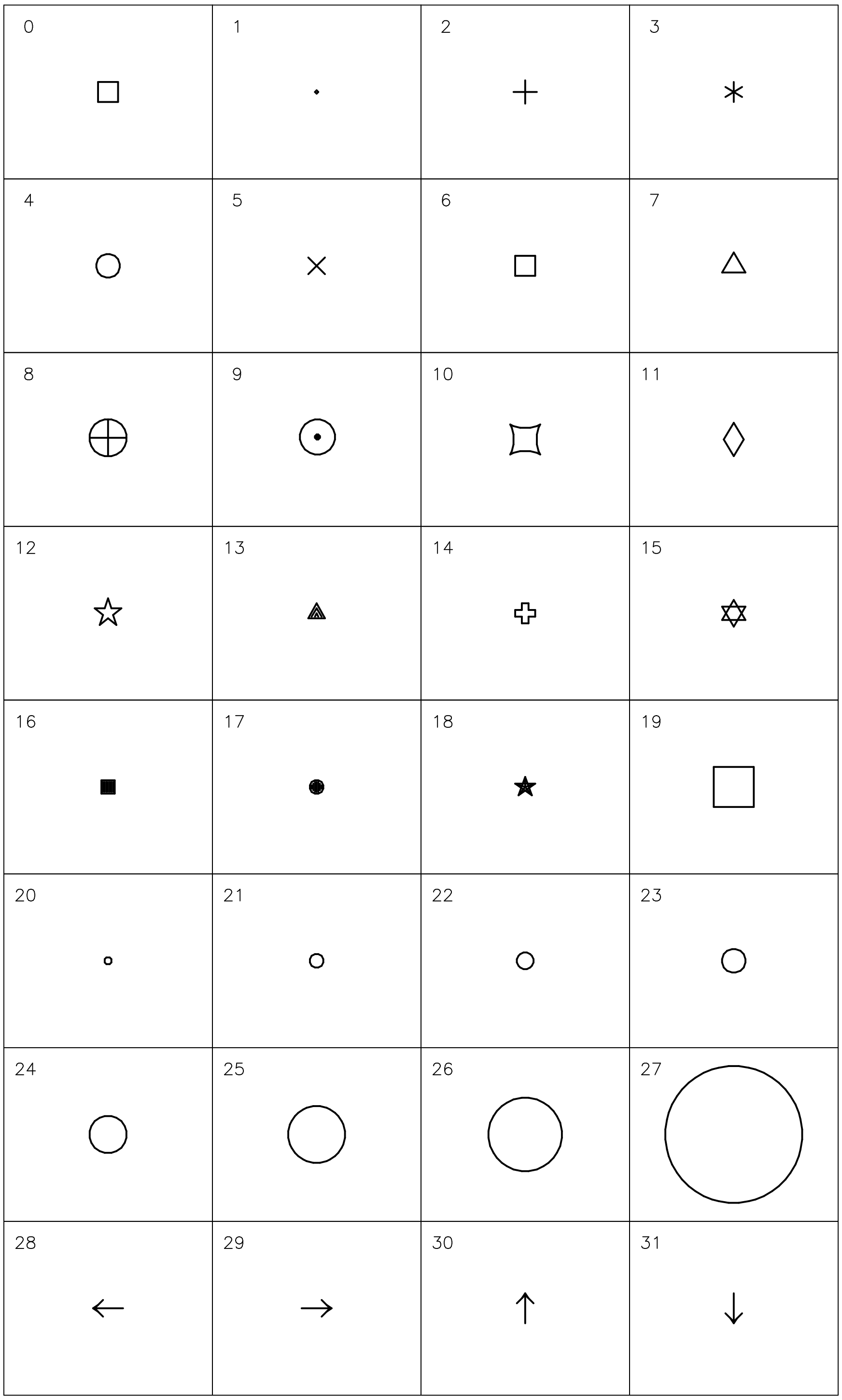

Figure 4.1 Standard Graph Markers.

4.5 Text

The Text primitive routine is used for writing labels and titles on the image. It converts an internal computer representation of the text (ASCII codes) into readable text. The simplest routine for writing text is `PGTEXT`, which writes a horizontal character string starting at a specific (x, y) world coordinate position, *e.g.*,

```
CALL PGTEXT (X, Y, 'A text string')
```

`PGTEXT` is actually a simplified interface to the more general primitive routine `PGPTEXT`, which allows one to change orientation and justification of the text, *e.g.*,

```
CALL PGPTEXT (X, Y, 45.0, 0.5, 'A text string')
```

writes the text at an angle of 45° to the horizontal, centered at (x, y) (see Appendix A).

Both `PGTEXT` and `PGMTEXT` require the position of the text string to be specified in world coordinates. When annotating a graph, it is usually more convenient to position the text relative to the edge of the viewport, rather than in world-coordinate space. The routine `PGMTEXT` (see Appendix A) is provided for this, and `PGLABEL` provides a simple interface to `PGMTEXT` for the normal job of annotating an (x, y) graph.

The appearance of text can be altered by specifying a number of *attributes*, described in the next chapter. In particular, the character size and character font can be changed. Figure 4.2 illustrates some of the possibilities.

To include one of the graph marker symbols in a text string, use the Fortran `CHAR` function, *e.g.*,

```
CALL PGTEXT (X, Y, 'Points marked with '//CHAR(17))
```

The routine `PGPTEXT` (and all the PGPLOT routines which call it, *e.g.*, `PGTEXT`, `PGLABEL`) allows one to include *escape sequences* in the text string to be plotted. These are character-sequences that are not plotted, but are interpreted as instructions to change font, draw superscripts or subscripts, draw non-ASCII characters (*e.g.*, Greek letters), etc. All escape sequences start with a backslash character (`\`). The following escape sequences are defined (the letter following the `\` may be either upper or lower case):

`\u` – start a superscript, or end a subscript;

`\d` – start a subscript, or end a superscript (note that `\u` and `\d` must always be used in pairs);

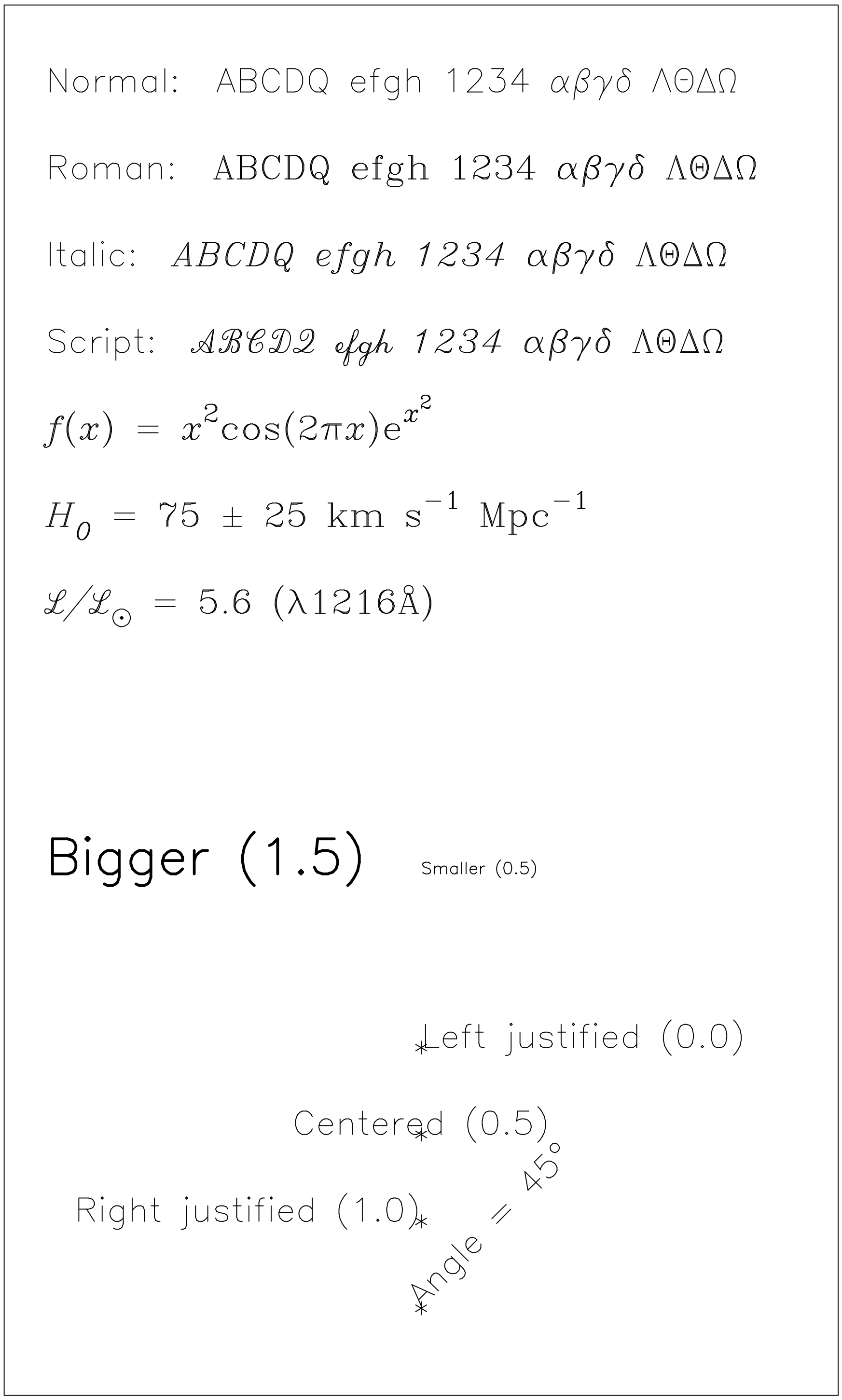

Figure 4.2 Text examples.

`\b` – backspace (*i.e.*, do not advance text pointer after plotting the previous character);

`\\` – backslash character (\);

`\A` – Ångström symbol (Å);

`\g`x – greek letter corresponding to roman letter x;

`\fn` – switch to Normal font (1);

`\fr` – switch to Roman font (2);

`\fi` – switch to Italic font (3);

`\fs` – switch to Script font (4);

`\(`n`)` – character number n (1 to 4 decimal digits); the closing parenthesis may be omitted if the next character is neither a digit nor ")". This makes a number of special characters (*e.g.*, mathematical, musical, astronomical, and cartographical symbols) available. See Appendix B for a list of available characters.

Greek letters are obtained by `\g` followed by one of the following upper-case and lower-case letters:

use:	A	B	G	D	E	Z	Y	H	I	K	L	M	N	C	O	P	R	S	T	U	F	X	Q	W
for:	A	B	$\varGamma$	$\varDelta$	E	Z	H	$\varTheta$	I	K	$\varLambda$	M	N	$\varXi$	O	$\varPi$	P	$\varSigma$	T	$\varUpsilon$	$\varPhi$	X	$\varPsi$	$\varOmega$
or:	a	b	g	d	e	z	y	h	i	k	l	m	n	c	o	p	r	s	t	u	f	x	q	w
for:	α	β	γ	δ	ϵ	ζ	η	θ	ι	κ	λ	μ	ν	ξ	o	π	ρ	σ	τ	υ	ϕ	χ	ψ	ω

Use uppercase letters for uppercase Greek, lowercase for lowercase. Example: `\gh` is θ, lowercase "theta".

4.6 Area Fill

The Area Fill primitive allows the programmer to shade the interior of an arbitrary polygonal region. The appearance of the primitive is controlled by attributes *fill area style* and *color index* (see Chapter 5). An area is specified by the set of vertices of the polygon.

The routine `PGPOLY` is used to fill an area. It has three arguments: the number (`N`) of vertices defining the polygon, and two arrays (`XPTS` and `YPTS`) containing the world x and y-coordinates of the vertices:

```
CALL PGPOLY (N, XPTS, YPTS)
```

If the polygon is not convex, it may not be obvious which points in the image are inside the polygon. PGPLOT assumes that a point is inside the polygon if a straight line drawn from the point to infinity intersects an odd number of the polygon's edges.

For the special case of a *rectangle* with edges parallel to the coordinates axes, it is better to use routine PGRECT instead of PGPOLY; this routine will use the hardware rectangle-fill capability if available. PGRECT has four arguments: the (x, y) world coordinates of two opposite corners (note the order of the arguments):

```
CALL PGRECT (X1, X2, Y1, Y2)
```

Chapter 5

ATTRIBUTES

5.1 Introduction

The appearance of the primitive elements of a graphical image (lines, graph-markers, text, and area-fill) can be changed by specifying *primitive attributes.* The attributes, and the corresponding routines for changing them, are:

Color Index and *Color Representation:* PGSCI, PGSCR, and PGSHLS.

Line Style: PGSLS.

Line Width: PGSLW.

Character Height: PGSCH.

Character Font: PGSCF.

Fill-area Style: PGSFS.

The routines to change attributes can be freely intermixed with the PGPLOT drawing routines. Once an attribute has been changed by a call to the appropriate routine, it remains in effect for all subsequent plotting until it is changed again. In addition to the routines that set attributes (PGSxx) there are routines for determining the current value of each attribute (PGQxx). These make it possible to write subroutines which change attribute values temporarily but restore the old attributes before returning to the calling program.

5.2 Color Index

This attribute affects all the primitives: lines, graph-markers, text, and area-fill, and is controlled by two subroutines: PGSCI and PGSCR.

Devices differ considerably in their ability to draw in more than one color. On most hardcopy devices, the default color is black on a white background, while on most CRT devices, it is white (or green) on a black background. Color is selected using an integer parameter called the *color index.* Color index 1 is the default color, and color index 0 is the background color. The number of different color indices available depends on the device. On most monochrome devices, only color indices 0 and 1 are available, while some color CRT devices may permit color indices from 0 to 255. On some monochrome devices, color index can be used to select different brightnesses (intensities).

Color index 0, the background color, can be used to "erase" elements from a picture by overwriting them with color index 0. Note that not all devices are capable of this: e.g., Tektronix storage-tube terminals and pen-plotters cannot erase part of a displayed picture.

To select a new color index for subsequent plotting, use routine `PGSCI` (Set Color Index).

Appendix D lists the capabilities of the devices for plotting in color and variable intensity. The default color index is 1; all devices accept this. Most devices also accept color index 0 (erase), and several accept color index up to 15 or more. The maximum color index is the number of different colors that can be displayed at once. Some devices permit the assignment of colors to color indices to be changed (by calling `PGSCR`, see below).

5.3 Color Representation

Each color index has an associated *Color Representation*, which defines the associated color and intensity. Color Representation may be expressed by a set of three numbers, either the Hue, Lightness, and Saturation (H, L, S) components or the Red, Green, and Blue (R, G, B) components. (R, G, B) are quantities in the range 0.0–1.0, with 1.0 being maximum intensity; if $R = G = B$ the color is a shade of gray. In the (H, L, S) system, hue is a cyclic quantity expressed as an angle in the range 0–360, while L and S are in the range 0.0–1.0.

Table 5.1 shows how the color indices are defined when PGPLOT is started (not all are available on all devices). The default assignments of colors to color indices can be changed with routine `PGSCR`, which permits one to specify the (R, G, B) values for any color index, or `PGSHLS`, which permits one to specify the (H, L, S) values. Note that color-index 0, the background color, can be redefined in this way.

The effect of `PGSCR` is device-dependent. One some devices, it will be ignored. On others, (*e.g.*, Grinnell, VT125) it will change the color of lines which have already been drawn in the specified color index, while on others (*e.g.*, pen plotters) it will only affect lines drawn after the call of `PGSCR`.

Table 5.1 Default Color Representations

Color Index	*Color*	(H, L, S)	(R, G, B)
0	Black (background)	0, 0.00, 0.00	0.00, 0.00, 0.00
1	White (default)	0, 1.00, 0.00	1.00, 1.00, 1.00
2	Red	120, 0.50, 1.00	1.00, 0.00, 0.00
3	Green	240, 0.50, 1.00	0.00, 1.00, 0.00
4	Blue	0, 0.50, 1.00	0.00, 0.00, 1.00
5	Cyan (Green + Blue)	300, 0.50, 1.00	0.00, 1.00, 1.00
6	Magenta (Red + Blue)	60, 0.50, 1.00	1.00, 0.00, 1.00
7	Yellow (Red + Green)	180, 0.50, 1.00	1.00, 1.00, 0.00
8	Red + Yellow (Orange)	150, 0.50, 1.00	1.00, 0.50, 0.00
9	Green + Yellow	210, 0.50, 1.00	0.50, 1.00, 0.00
10	Green + Cyan	270, 0.50, 1.00	0.00, 1.00, 0.50
11	Blue + Cyan	330, 0.50, 1.00	0.00, 0.50, 1.00
12	Blue + Magenta	30, 0.50, 1.00	0.50, 0.00, 1.00
13	Red + Magenta	90, 0.50, 1.00	1.00, 0.00, 0.50
14	Dark Gray	0, 0.33, 0.00	0.33, 0.33, 0.33
15	Light Gray	0, 0.66, 0.00	0.66, 0.66, 0.66
16–255	Undefined		

5.4 Line Style

Line Style can be, *e.g.*, solid, dashed, or dotted. The attribute affects only lines, not the other primitives. It is controlled by subroutine `PGSLS`. The default line style is a full, unbroken line. To change the line style, use routine `PGSLS`. Line style is described by an integer code:

1 – full line,

2 – long dashes,

3 – dash-dot-dash-dot,

4 – dotted,

5 – dash-dot-dot-dot.

5.5 Line Width

Line Width affects lines, graph-markers, and text. A thick-nibbed pen is simulated by tracing each line with multiple strokes offset in the direction perpendicular to the line. The line width is specified by the number of strokes. The default width is one stroke, and the maximum that may be specified is 201. The exact appearance of thick lines is device-dependent—it depends on the resolution of the device—but on hardcopy devices (*e.g.*, QMS Lasergrafix, Versatec) PGPLOT attempts to make the line-width increment equal to 0.005 inches. Requesting a line-width of 10, say, should give lines that are approximately 1/20 inch thick. To change the line width, use routine `PGSLW`.

5.6 Character Height

Character Height affects graph-markers and text. Character height is specified as a multiple of the default character height; the default character height one-fortieth of the height or width of the view surface (whichever is less). To change the character height, use routine `PGSCH`.

5.7 Character Font

Character Font affects text only. Four fonts are available. The default font (1) is simple and is the fastest to draw; the others should only be used for presentation plots on a high-resolution device (*e.g.*, Versatec or laser printer). To change the character font, use routine `PGSCF`; it is also possible to change the font temporarily by using escape sequences (see §4.4). The font is defined by an integer code:

1 – normal (simple) font (default),

2 – roman font,

3 – italic font,

4 – script font.

5.8 Fill-Area Style

Fill-Area Style can be hollow (only the outline of the polygon is drawn), or solid. The attribute may be extended in future to allow hatching and other patterns. To change the fill-are style, use routine `PGSFS`. The style is defined by an integer code:

1 – solid fill (default),

2 – hollow (outline only).

Chapter 6

HIGHER-LEVEL ROUTINES

6.1 Introduction

This chapter describes a number of "high level" routines that simplify the composition of complicated graphical images. They include routines for drawing graphs of one variable or function against another ("xy-plots"), histograms, and display of two-dimensional data (functions of two variables). Rather than giving complete details of all the available routines, this chapter just points out some of the ways that they can be used. See Appendix A for details of the calling sequences.

6.2 XY-plots

The basic technique for drawing xy-plots is described in Chapter 2, which showed how to make *scatter plots* using graph markers produced by PGPOINT and *line plots* produced by PGLINE. Considerable variation in the appearance of the graph can be achieved using the following techniques.

Attributes. Use different attributes to distinguish different datasets. Graph markers can be distinguished by choosing different markers, different colors, or different sizes (character height attribute). Lines and curves can be distinguished by line-style, color, or line-width.

Box parameters. If routine PGENV is replaced by calls to the more basic routines (see §3.6), including PGBOX, considerable variety in the appearance of the graph can be achieved. For example, one can suppress the tick marks, draw the tick marks projecting out of the box instead of into it, or draw a grid over the whole viewport. Note that PGBOX may be called many times: one might call it once to draw a grid using thin dotted lines, and again to draw the frame and tick marks using thick lines:

```
CALL PGSLW(1)
CALL PGSLS(4)
CALL PGBOX('G',30.0,0,'G',0.2,0)
CALL PGSLW(3)
CALL PGSLS(1)
CALL PGBOX('ABCTSN',90.0,3,'ABCTSNV',0.0,0)
```

Note that in this example we have also specified tick intervals explicitly. If the horizontal axis is to represent an angle in degrees, it is convenient to choose a tick interval that is a simple fraction of 360; here we have a major tick interval of 90° and a minor tick interval of 30°.

Stepped-line plots. As an alternative to `PGLINE`, which "joins up the dots" using straight line segments, it is sometimes appropriate to use `PGBIN` which produces a "stepped line plot" (sometimes misleadingly called a histogram) with horizontal line segments at each data point and vertical line segments joining them. This is often used, for example, in displaying digitized spectra.

Error bars. Graphs of real data often require the inclusion of error bars. The two routines `PGERRX` and `PGERRY` draw horizontal and vertical error bars, respectively. These routines are usually used in combination with `PGPOINT`, *e.g.*, to draw a set of points with $\pm 2\sigma$ error-bars:

```
      DO 10 I=1,15
          YHI = YPTS(I) + 2.0*ERR(I)
          YLO = YPTS(I) - 2.0*ERR(I)
          CALL PGPOINT(1, XPTS(I), YPTS(I), 17)
          CALL PGERRY(1, XPTS(I), YLO, YHI, 1.0)
   10 CONTINUE
```

Logarithmic axes. It is commonly required that the x-axis, the y-axis, or both, be logarithmic instead of linear; that is, one wishes to plot the logarithm of the quantity instead of its actual value. PGPLOT doesn't provide any automatic mechanism to do this: one has to adopt $\log_{10} x$ and/or $\log_{10} y$ instead of x and y as world-coordinates; *i.e.*, if the range of x is to be 1 to 1000, choose as world-coordinate limits for the window $\log 1 = 0.0$ and $\log 1000 = 3.0$, and supply the logarithms of x to `PGPOINT` and `PGLINE`. However, `PGENV` and `PGBOX` have options for *labeling* the axis logarithmically; if this option is used in our example, the axis will have labeled major tick marks at 1, 10, 100, and 1000, with logarithmically-spaced minor tick marks at 2, 3, 4, ..., 20, 30, 40, etc.. An example may make this clearer:

```
      CALL PGENV(-2.0,2.0,-0.5,2.5,1,30)
      CALL PGLABEL('Frequency, \gn (GHz)',
     1             'Flux Density, S\d\gn\u (Jy)', ' ')
      DO 10 I=1,15
          XPTS(I) = ALOG10(FREQ(I))
          YPTS(I) = ALOG10(FLUX(I))
   10 CONTINUE
      CALL PGPOINT(15, XPTS, YPTS, 17)
```

This is a fragment of a program to draw the spectrum of a radio source, which is usually plotted as a log–log plot of flux density *v.* frequency. It first calls `PGENV` to initialize the viewport and window; the `AXIS` argument is 30 so both axes will be logarithmic. The x-axis (frequency, ν) runs from 0.01 to 100 GHz, the y-axis (flux density, S_ν) runs from 0.3 to 300 Jy. Note that it is necessary to specify the logarithms of these limits in the call to `PGENV`. The penultimate argument requests equal scales in x and y so that slopes will be correct. The program then marks 15 data points, supplying the *logarithms* of frequency and flux density to `PGPOINT`.

6.3 Histograms

The routine `PGHIST` draws a histogram, that is, the frequency distribution of measured values in a dataset. Suppose we have 500 measurements of a quantity (the sky brightness temperature at 20 GHz, say, in mK) stored in Fortran array `VALUES`. The following program-fragment draws a histogram of the distribution of these values in the range 0.0 to 5.0, using 25 bins (so that each bin is 0.2 K wide, the first running from 0.0 to 0.2, the second from 0.2 to 0.4, etc.):

```
      DO 10 I=1,500
          VALUES(I) = ....
   10 CONTINUE
      CALL PGHIST(500, VALUES, 0.00, 5.00, 25, 0)
      CALL PGLABEL('Temperature (K)',
     1             'Number of measurements',
     2             'Sky Brightness at 20 GHz' )
```

The histogram does not depend on the *order* of the values within the array.

6.4 Functions of two variables

A function of two variables, $f = f(x, y)$, really needs a three-dimensional display. `PGPLOT` does not have any three-dimensional display capability, but it provides three methods for two-dimensional display of three-dimensional data.

Contour maps. In a contour map of $f(x,y)$, the world-coordinates are x and y and the contours are lines of constant f. The PGPLOT contouring routines (`PGCONT` and `PGCONS`) require the input data to be stored in a two-dimensional Fortran array `F`, with element `F(I,J)` containing the value of the function $f(x,y)$ for a point (x_i, y_j). Furthermore, the function must be sampled on a regular grid: the (x,y) coordinates corresponding to `(I,J)` must be related to `I` and `J` by:

$$x = a + b\mathtt{I} + c\mathtt{J},$$
$$y = d + e\mathtt{I} + f\mathtt{J}.$$

The constants a, b, c, d, e, f are supplied to `PGCONT` in a six-element Fortran array. The other input required is an array containing the contour values, *i.e.*, the constant values of f corresponding to each contour to be drawn. In the following example, we assume that values have been assigned to the elements of array `F`. We first find the maximum and minimum values of `F`, and choose 10 contour levels evenly spaced between the maximum and minimum:

```
      REAL F(50,50), ALEV(10), TR(6)
      ...
      FMIN = F(1,1)
      FMAX = F(1,1)
      DO 300 I=1,50
        DO 200 J=1,50
          FMIN = MIN(F(I,J),FMIN)
          FMAX = MAX(F(I,J),FMAX)
  200   CONTINUE
  300 CONTINUE
      DO 400 I=1,10
        ALEV(I) = FMIN + (I-1)*(FMAX-FMIN)/9.0
  400 CONTINUE
```

Next, we choose a window and viewport, and set up an array `TR` containing the 6 constants in the transformation between array indices and world coordinates. In this case, the transformation is simple, as we want $x =$ `I`, $y =$ `J`:

```
      CALL PGENV(0.,50.,5.,45.,0,2)
      TR(1) = 0.0
      TR(2) = 1.0
      TR(3) = 0.0
      TR(4) = 0.0
      TR(5) = 0.0
      TR(6) = 1.0
```

Finally, we call PGCONT; actually, we call it twice, to draw the first five contours in color index 2 (red) and the remaining 5 in color index 3 (green):

```
CALL PGSCI(2)
CALL PGCONT(F,50,50,1,50,1,50,ALEV,5,TR)
CALL PGSCI(3)
CALL PGCONT(F,50,50,1,50,1,50,ALEV(6),5,TR)
```

Normally PGCONT is preferable to PGCONS. See the description in Appendix A for suggestions as to when PGCONS should be used.

Gray-scale plots. The routine PGGRAY is used in a similar way to PGCONT. Instead of drawing contours, it shades the interior of the viewport, the intensity of shading representing the value of the function at each point. The exact appearance of the resulting "image" is device-dependent. On some devices, PGGRAY does the shading by drawing many dots, so it can be *very* slow.

Cross sections. Routine PGHI2D draws a series of cross-sections through a two-dimensional data array. Each cross-section "hides" those that appear behind it, giving a three-dimensional effect. See Appendix A for details.

Chapter 7

INTERACTIVE GRAPHICS

7.1 Introduction

The previous chapters have described how to produce a *static* graphical image: if the same program is run twice with the same input parameters, the same image will result. An *interactive* program allows the user to control the behavior of the program with a graphical input device. PGPLOT supports a limited interactive capability on devices with a cursor for graphical input (*e.g.*, Grinnell, VT125 terminal, some Tektronix terminals, VT640 Retrographics terminal). The capabilities are necessarily limited by the aim to keep PGPLOT device-independent.

7.2 The Cursor

Some of the graphics devices supported by PGPLOT have a *graphics cursor*. This appears on the view surface as a plus sign, a cross-hair, or a diamond, and can be moved around the view surface with a joy-stick, mouse, or trackball attached to the graphics device. If the hardware does not provide this mechanism, PGPLOT allows the user to move the cursor using the arrow keys on his terminal. See Appendix D for instructions for using the cursor on a specific device.

7.3 Using the Cursor

The basic routine for cursor input is `PGCURSE`. This routine enables the cursor on the selected device, positions it at a specified location within the viewport, and allows the user to move it. When the user has positioned the cursor, he types a key on his terminal; `PGCURSE` returns the cursor position (in world coordinates) and the character that was typed.

In addition, PGPLOT provides three higher-level routines for cursor input: `PGOLIN`, `PGNCURSE`, and `PGLCUR`. These three routines require that the device has erase capability.

PGOLIN allows the user to specify a set of points within the viewport, with the capability of correcting mistakes. Interactive commands (single characters [A, D, or X] typed on the keyboard) allow the user to *add* a point at the current cursor position, *delete* the last-entered point, or *exit* from the subroutine. The world-coordinates of the entered points are returned to the calling program. The following program fragment illustrates the use of PGOLIN; the user supplies NPT (up to 50) points with world-coordinates X() and Y(), and the program then shades the polygon defined by these points by calling PGPOLY:

```
INTEGER NPT
REAL X(50), Y(50)
...
WRITE (6,*) 'Use the cursor to draw a polygon'
WRITE (6,*) 'Type A to add point, D to delete, X to exit'
NPT = 0
CALL PGOLIN (50, NPT, X, Y, 0)
IF (NPT.GE.3) CALL PGPOLY (NPT, X, Y)
```

PGNCURSE is similar to PGOLIN, but the points are sorted into increasing order of x before being returned to the calling program. In addition, the *delete* command deletes the point nearest to the cursor, rather than the last-entered point. It can be used, for example, to allow the user to supply a set of points to represent the continuum level on a spectrum.

PGLCUR is similar to PGOLIN but instead of using a graph marker to mark each entered point it draws a polyline through them.

7.4 Buffering

By default, PGPLOT ensures that the image seen on the view surface is up to date at all times; that is, each PGPLOT subroutine updates the image before returning control to the calling program. To improve efficiency, PGPLOT can save instructions for controlling the graphics device in a buffer, and only send them to the device when the buffer is filled up. This means that at any given moment, the image displayed on the screen may not be completely up to date. This can be a problem in an interactive program, where, for example, the user has to tell the program what to do next based on his interpretation of the current display. Three PGPLOT routines (PGBBUF, PGEBUF, and PGUPDT) are provided for controlling the buffering of output. All three routines have no arguments.

The routine PGBBUF causes PGPLOT to begin saving graphical output in a buffer. The output is saved until (1) a matching PGEBUF call is made, or (2) the buffer fills up, or (3) the buffer is emptied by a call to PGUPDT, or (4) PGEND is called. The routine PGEBUF stops buffering and causes the buffered commands to be sent to the output device. Calls to PGBBUF and PGEBUF should always be paired. PGBBUF increments an internal counter, while PGEBUF decrements this counter and flushes the buffer to the output device when the counter drops to zero. This allows a subroutine to turn on and turn off buffering without disturbing any buffering that may have been established by the calling program.

Routine PGUPDT empties the buffer created by PGBBUF, but it does not alter the internal counter. The routine should be called when it is essential that the display be completely up-to-date (before interaction with the user, for example) but it is not known if output is being buffered.

Usually output is not buffered; this is the default state established by PGBEGIN. The default behavior can be changed, however, by defining an environment variable PGPLOT_BUFFER (see Chapter 1). If this variable is defined, with any value, PGBEGIN will start buffering output (by calling PGBBUF).

The following example shows how routine PGLABEL might be implemented in terms of routine PGMTEXT:

```
SUBROUTINE PGLABEL (XLBL, YLBL, TOPLBL)
CHARACTER*(*) XLBL, YLBL, TOPLBL
CALL PGBBUF
CALL PGMTEXT('T', 2.0, 0.5, 0.5, TOPLBL)
CALL PGMTEXT('B', 3.2, 0.5, 0.5, XLBL)
CALL PGMTEXT('L', 2.2, 0.5, 0.5, YLBL)
CALL PGEBUF
END
```

The calls to PGBBUF and PGEBUF ensure that the output generated by the three calls to PGMTEXT is buffered (i.e., sent to the output device as a single command instead of three separate ones). If buffering is already enabled by the program which calls PGLABEL, the calls to PGBBUF and PGEBUF have no effect.

Chapter 8

METAFILES

8.1 Introduction

A graphics *metafile* is a disk file in which a device-independent representation of a graphics image can be stored. Such a file cannot be displayed directly on a graphics device, but must first be translated into the commands appropriate for the specific device using a *metafile translator*. This may seem like an unnecessary complication, when PGPLOT can create the device-specific commands directly, but it has some advantages. Metafiles can be used, for example, to transfer pictures between two computing sites (they are usually smaller than the corresponding device-specific files) or for archiving pictures. It is sometimes convenient to make a program generate a metafile rather than a device-specific file so that one can, for example, take a "quick look" at the picture on an interactive display before making hard copies. One may not know what hard-copy devices will be available when the program runs: for example, if it turns out after your 6-hour batch job has finished that the Versatec printer has broken, it is nice to be able to send the plot to some other device without re-running the batch job.

The metafiles generated by PGPLOT follow the "GSPC Metafile Proposal" described in *Computer Graphics (A. C. M.)*, volume **13**, number 3 (August 1979).

At present, metafiles are available only in the VMS version of PGPLOT, not in the UNIX versions.

8.2 Creating Metafiles

Metafiles may be created using PGPLOT in the same way that any other plot file is created: the device specification consists of a disk file name and a device type `/FILE`, for example `PLOT17.GMF/FILE`. The default file type if none is specified is `.GMF` (for Graphics Meta File). Programs which might generate metafile output should not make any assumptions about the physical scale of the picture; the scale will vary depending on what device the picture is ultimately plotted on.

8.3 Translating Metafiles

In order to generate graphics output from a metafile, one must use a *Metafile Translator* to interpret the device independent metafile commands. The Metafile Translator (GMFPLOT) provided with PGPLOT uses the PGPLOT subroutines to generate the device-specific output: thus a metafile may be displayed on any of the devices supported by PGPLOT. (A metafile may even be translated into another metafile, but this is not very useful.) To use GMFPLOT, first define a command `PLOT`, say, as follows; this definition may be included in your `LOGIN.COM` file if you make extensive use of it:

```
$ PLOT == "$PGPLOT_DIR:GMFPLOT"
```

The `PLOT` command takes two arguments: the name of the input metafile, and the device specification for the output. The following sample commands display a metafile on a Grinnell and on a Versatec printer:

```
$ PLOT PLOT17.GMF /GR
$ PLOT PLOT17 DEIMOS::LVA0:/VE
```

Again, the default file type for the metafile is `.GMF`.

At present, it is not possible to edit the metafile before display. This is a facility which might be added one day. Nor is it possible to change the scale or *aspect ratio* (ratio of height/width of the display surface). When PGPLOT generates a metafile, it does not know the size or shape of the display surface that it will ultimately be plotted on; as it has to make some assumption, it assumes that the surface will be square. The metafile translator displays the metafile in the largest square available on the output device. Thus a plot which is sent directly to a terminal may not look exactly the same as one that is stored in a metafile and subsequently displayed on the terminal. Future enhancements may allow one to specify the scale and aspect ratio of the metafile when it is generated.

Appendix A

SUBROUTINE DESCRIPTIONS

A.1 Introduction

This appendix includes a classified list of all the PGPLOT subroutines, and then gives detailed instructions for the use of each routine in Fortran programs. The subroutine descriptions are in alphabetical order.

A.2 Arguments

The subroutine descriptions indicate the data type of each argument. When arguments are described as "input", they may be replaced with constants or expressions in the `CALL` statement, but make sure that the constant or expression has the correct data type.

1. `INTEGER` arguments: these should be declared `INTEGER` or `INTEGER*4` in the calling program, not `INTEGER*2`.
2. `REAL` arguments: these should be declared `REAL` or `REAL*4` in the calling program, not `REAL*8` or `DOUBLE PRECISION`.
3. `CHARACTER` arguments: any valid Fortran `CHARACTER` variable may be used (declared `CHARACTER*`n for some integer n).

A.3 Classified List

Note: all routine names begin with the letters "PG". Most (but unfortunately not all) routine names are six characters or less, to conform to Fortran-77 standards.

Control routines

PGADVANCE – see PGPAGE

PGASK – control new page prompting

PGBBUF – begin batch of output (buffer)

PGBEGIN – begin PGPLOT, open output device

PGEBUF – end batch of output (buffer)

PGEND – terminate PGPLOT

PGPAGE – advance to a new page or clear screen

PGPAPER – change the size of the view surface

PGUPDT – update display

Windows and viewports

PGBOX – draw labeled frame around viewport

PGENV – set window and viewport and draw labeled frame

PGVPORT – set viewport (normalized device coordinates)

PGVSIZE – set viewport (inches)

PGVSTAND – set standard (default) viewport

PGWINDOW – set window

PGWNAD – set window and adjust viewport to same aspect ratio

Primitive drawing routines

PGDRAW – draw a line from the current pen position to a point

PGLINE – draw a polyline (curve defined by line-segments)

PGMOVE – move pen (change current pen position)

PGPOINT – draw one or more graph markers

PGPOLY – fill a polygonal area with shading

PGRECT – draw a rectangle, using fill-area attributes

Text

PGLABEL – write labels for x-axis, y-axis, and top of plot

PGMTEXT – write text at position relative to viewport

PGPTEXT – write text at arbitrary position and angle

PGTEXT – write text (horizontal, left-justified)

Attribute setting

PGSCF – set character font

PGSCH – set character height

PGSCI – set color index

PGSCR – set color representation

PGSFS – set fill-area style

PGSHLS – set color representation using HLS system

PGSLS – set line style

PGSLW – set line width

Higher-level drawing routines

PGBIN – histogram of binned data

PGCONS – contour map of a 2D data array (fast algorithm)

PGCONT – contour map of a 2D data array (contour-following)

PGCONX – contour map of a 2D data array (non-rectangular)

PGERRX – horizontal error bar

PGERRY – vertical error bar

PGFUNT – function defined by X = F(T), Y = G(T)

PGFUNX – function defined by Y = F(X)

PGFUNY – function defined by X = F(Y)

PGGRAY – gray-scale map of a 2D data array

PGHI2D – cross-sections through a 2D data array

PGHIST – histogram of unbinned data

Interactive graphics (cursor)

PGCURSE – read cursor position

PGLCUR – draw a line using the cursor

PGNCURSE – mark a set of points using the cursor

PGOLIN – mark a set of points using the cursor

Inquiry routines

PGQCF – inquire character font

PGQCH – inquire character height

PGQCI – inquire color index

PGQCR – inquire color representation

PGQFS – inquire fill-area style

PGQINF – inquire PGPLOT general information

PGQLS – inquire line style

PGQLW – inquire line width

PGQVP – inquire viewport size and position

PGQWIN – inquire window boundary coordinates

Utility routines

PGETXT – erase text from graphics display

PGIDEN – write username, date, and time at bottom of plot

PGLDEV – list available device types

PGNUMB – convert a number into a plottable character string

PGRND – find the smallest "round" number greater than x

PGRNGE – choose axis limits

A.4 Subroutine Synopses

The following pages give descriptions of all the PGPLOT subroutines in alphabetical order. These descriptions have been extracted from comments in the Fortran source code. (For an up-to-date version of these decriptions, look at the file `PGPLOT.DOC` in the PGPLOT directory.)

A.5 PGADVANCE – non-standard alias for PGPAGE

```
      SUBROUTINE PGADVANCE
```

A.6 PGASK – control new page prompting

```
      SUBROUTINE PGASK (FLAG)
      LOGICAL FLAG
Change the "prompt state" of PGPLOT. If the prompt state is
ON, PGPAGE will type "Type <RETURN> for next page:" and will wait
for the user to type <CR> before starting a new page.  The initial
prompt state (after a call to PGBEG) is ON for interactive devices.
Prompt state is always OFF for non-interactive devices.
Arguments:
 FLAG   (input)  : if .TRUE., and if the device is an interactive
                   device, the prompt state will be set to ON. If
                   .FALSE., the prompt state will be set to OFF.
```

A.7 PGBBUF – begin batch of output (buffer)

```
      SUBROUTINE PGBBUF
Begin saving graphical output commands in an internal buffer; the
commands are held until a matching PGEBUF call (or until the buffer
is emptied by PGUPDT). This can greatly improve the efficiency of
PGPLOT.  PGBBUF increments an internal counter, while PGEBUF
decrements this counter and flushes the buffer to the output
device when the counter drops to zero.  PGBBUF and PGEBUF calls
should always be paired.
Arguments: none
```

A.8 PGBEG – begin PGPLOT, open output device

```
      INTEGER FUNCTION PGBEG (UNIT, FILE, NXSUB, NYSUB)
      INTEGER       UNIT
      CHARACTER*(*) FILE
      INTEGER       NXSUB, NYSUB
Begin PGPLOT, open the plot file.  A call to PGBEG is
required before any other calls to PGPLOT subroutines.  If a plot
file is already open for PGPLOT output, it is closed before the new
file is opened.
Returns:
 PGBEG         : a status return value. A value of 1 indicates
                   successful completion, any other value indicates
                   an error. In the event of error a message is
                   written on the standard error unit.
                   To test the return value, call
                   PGBEG as a function, eg IER=PGBEG(...); note
                   that PGBEG must be declared INTEGER in the
                   calling program.
Arguments:
 UNIT  (input)   : this argument is ignored by PGBEG (use zero).
 FILE  (input)   : the "device specification" for the plot device.
                   Device specifications are installation dependent,
                   but usually have the form "device/type" or
                   "file/type". If this argument is a
                   question mark ('?'), PGBEG will prompt the user
                   to supply a string.
 NXSUB  (input)  : the number of subdivisions of the view surface in
                   X.
 NYSUB  (input)  : the number of subdivisions of the view surface in
                   Y. PGPLOT puts NXSUB x NYSUB graphs on each plot
                   page or screen; when the view surface is sub-
                   divided in this way, PGPAGE moves to the next
                   sub-page, not the  next physical page.
```

A.9 PGBEGIN – non-standard alias for PGBEG

```
      INTEGER FUNCTION PGBEGIN (UNIT, FILE, NXSUB, NYSUB)
      INTEGER       UNIT
      CHARACTER*(*) FILE
      INTEGER       NXSUB, NYSUB
```

A.10 PGBIN – histogram of binned data

```
      SUBROUTINE PGBIN (NBIN, X, DATA, CENTER)
      INTEGER NBIN
      REAL X(*), DATA(*)
      LOGICAL CENTER
Plot a histogram of NBIN values with X(1..NBIN) values along
the ordinate, and DATA(1...NBIN) along the abscissa. Bin width is
spacing between X values.
Arguments:
 NBIN   (input)  : number of values.
 X      (input)  : abscissae of bins.
 DATA   (input)  : data values of bins.
 CENTER (input)  : if .TRUE., the X values denote the center of the
                   bin; if .FALSE., the X values denote the lower
                   edge (in X) of the bin.
```

A.11 PGBOX – draw labeled frame around viewport

```
      SUBROUTINE PGBOX (XOPT, XTICK, NXSUB, YOPT, YTICK, NYSUB)
      CHARACTER*(*) XOPT, YOPT
      REAL XTICK, YTICK
      INTEGER NXSUB, NYSUB
Annotate the viewport with frame, axes, numeric labels, etc.
PGBOX is called by on the user's behalf by PGENV, but may also be
called explicitly.
Arguments:
 XOPT   (input)  : string of options for X (horizontal) axis of
                   plot. Options are single letters, and may be in
                   any order (see below).
 XTICK  (input)  : world coordinate interval between major tick marks
                   on X axis. If XTICK=0.0, the interval is chosen by
                   PGBOX, so that there will be at least 3 major tick
                   marks along the axis.
 NXSUB  (input)  : the number of subintervals to divide the major
                   coordinate interval into. If XTICK=0.0 or NXSUB=0,
                   the number is chosen by PGBOX.
 YOPT   (input)  : string of options for Y (vertical) axis of plot.
                   Coding is the same as for XOPT.
 YTICK  (input)  : like XTICK for the Y axis.
 NYSUB  (input)  : like NXSUB for the Y axis.
Options (for parameters XOPT and YOPT):
 A : draw Axis (X axis is horizontal line Y=0, Y axis is vertical
     line X=0).
 B : draw bottom (X) or left (Y) edge of frame.
 C : draw top (X) or right (Y) edge of frame.
 G : draw Grid of vertical (X) or horizontal (Y) lines.
 I : Invert the tick marks; ie draw them outside the viewport
     instead of inside.
 L : label axis Logarithmically (see below).
 N : write Numeric labels in the conventional location below the
     viewport (X) or to the left of the viewport (Y).
 P : extend ("Project") major tick marks outside the box (ignored if
     option I is specified).
 M : write numeric labels in the unconventional location above the
     viewport (X) or to the right of the viewport (Y).
 T : draw major Tick marks at the major coordinate interval.
 S : draw minor tick marks (Subticks).
 V : orient numeric labels Vertically. This is only applicable to Y.
     The default is to write Y-labels parallel to the axis
To get a complete frame, specify BC in both XOPT and YOPT.
Tick marks, if requested, are drawn on the axes or frame
or both, depending which are requested. If none of ABC is specified,
tick marks will not be drawn. When PGENV calls PGBOX, it sets both
XOPT and YOPT according to the value of its parameter AXIS:
-1: 'BC', 0: 'BCNST', 1: 'ABCNST', 2: 'ABCGNST'.
For a logarithmic axis, the major tick interval is always 1.0. The
numeric label is 10**(x) where x is the world coordinate at the
tick mark. If subticks are requested, 8 subticks are drawn between
each major tick at equal logarithmic intervals.
```

A.12 PGCONB – contour map of a 2D data array, with blanking

```
      SUBROUTINE PGCONB (A, IDIM, JDIM, I1, I2, J1, J2, C, NC, TR,
     1                   BLANK)
      INTEGER IDIM, JDIM, I1, I2, J1, J2, NC
      REAL    A(IDIM,JDIM), C(*), TR(6), BLANK
Draw a contour map of an array. This routine is the same as PGCONS,
except that array elements that have the "magic value" defined by
argument BLANK are ignored, making gaps in the contour map. The
routine may be useful for data measured on most but not all of the
points of a grid.
Arguments:
 A      (input)  : data array.
 IDIM   (input)  : first dimension of A.
 JDIM   (input)  : second dimension of A.
 I1,I2  (input)  : range of first index to be contoured (inclusive).
 J1,J2  (input)  : range of second index to be contoured (inclusive).
 C      (input)  : array of contour levels (in the same units as the
                   data in array A); dimension at least NC.
 NC     (input)  : number of contour levels (less than or equal to
                   dimension of C). The absolute value of this
                   argument is used (for compatibility with PGCONT,
                   where the sign of NC is significant).
 TR     (input)  : array defining a transformation between the I,J
                   grid of the array and the world coordinates. The
                   world coordinates of the array point A(I,J) are
                   given by:
                     X = TR(1) + TR(2)*I + TR(3)*J
                     Y = TR(4) + TR(5)*I + TR(6)*J
                   Usually TR(3) and TR(5) are zero - unless the
                   coordinate transformation involves a rotation
                   or shear.
 BLANK   (input) : elements of array A that are exactly equal to
                   this value are ignored (blanked).
```

A.13 PGCONS – contour map of a 2D data array (fast algorithm)

```
      SUBROUTINE PGCONS (A, IDIM, JDIM, I1, I2, J1, J2, C, NC, TR)
      INTEGER IDIM, JDIM, I1, I2, J1, J2, NC
      REAL    A(IDIM,JDIM), C(*), TR(6)
```

```
Draw a contour map of an array. The map is truncated if
necessary at the boundaries of the viewport.  Each contour line is
drawn with the current line attributes (color index, style, and
width).  This routine, unlike PGCONT, does not draw each contour as a
continuous line, but draws the straight line segments composing each
contour in a random order.  It is thus not suitable for use on pen
plotters, and it usually gives unsatisfactory results with dashed or
dotted lines.  It is, however, faster than PGCONT, especially if
several contour levels are drawn with one call of PGCONS.
Arguments:
 A      (input)  : data array.
 IDIM   (input)  : first dimension of A.
 JDIM   (input)  : second dimension of A.
 I1,I2  (input)  : range of first index to be contoured (inclusive).
 J1,J2  (input)  : range of second index to be contoured (inclusive).
 C      (input)  : array of contour levels (in the same units as the
                   data in array A); dimension at least NC.
 NC     (input)  : number of contour levels (less than or equal to
                   dimension of C). The absolute value of this
                   argument is used (for compatibility with PGCONT,
                   where the sign of NC is significant).
 TR     (input)  : array defining a transformation between the I,J
                   grid of the array and the world coordinates. The
                   world coordinates of the array point A(I,J) are
                   given by:
                     X = TR(1) + TR(2)*I + TR(3)*J
                     Y = TR(4) + TR(5)*I + TR(6)*J
                   Usually TR(3) and TR(5) are zero - unless the
                   coordinate transformation involves a rotation
                   or shear.
```

A.14 PGCONT – contour map of a 2D data array (contour-following)

```
      SUBROUTINE PGCONT (A, IDIM, JDIM, I1, I2, J1, J2, C, NC, TR)
      INTEGER IDIM, JDIM, I1, J1, I2, J2, NC
      REAL A(IDIM,JDIM), C(*), TR(6)
Draw a contour map of an array.  The map is truncated if
necessary at the boundaries of the viewport.  Each contour line
is drawn with the current line attributes (color index, style, and
width); except that if argument NC is positive (see below), the line
style is set by PGCONT to 1 (solid) for positive contours or 2
(dashed) for negative contours.
Arguments:
 A      (input) : data array.
 IDIM   (input) : first dimension of A.
 JDIM   (input) : second dimension of A.
 I1, I2 (input) : range of first index to be contoured (inclusive).
 J1, J2 (input) : range of second index to be contoured (inclusive).
 C      (input) : array of NC contour levels; dimension at least NC.
 NC     (input) : +/- number of contour levels (less than or equal
                  to dimension of C). If NC is positive, it is the
                  number of contour levels, and the line-style is
                  chosen automatically as described above. If NC is
                  negative, it is minus the number of contour
                  levels, and the current setting of line-style is
                  used for all the contours.
 TR     (input) : array defining a transformation between the I,J
                  grid of the array and the world coordinates.
                  The world coordinates of the array point A(I,J)
                  are given by:
                    X = TR(1) + TR(2)*I + TR(3)*J
                    Y = TR(4) + TR(5)*I + TR(6)*J
                  Usually TR(3) and TR(5) are zero - unless the
                  coordinate transformation involves a rotation or
                  shear.
```

A.15 PGCONX – contour map of a 2D data array (non-rectangular)

```
      SUBROUTINE PGCONX (A, IDIM, JDIM, I1, I2, J1, J2, C, NC, PLOT)
      INTEGER  IDIM, JDIM, I1, J1, I2, J2, NC
      REAL     A(IDIM,JDIM), C(*)
      EXTERNAL PLOT
Draw a contour map of an array using a user-supplied plotting
routine.  This routine should be used instead of PGCONT when the
data are defined on a non-rectangular grid.  PGCONT permits only
a linear transformation between the (I,J) grid of the array
and the world coordinate system (x,y), but PGCONX permits any
transformation to be used, the transformation being defined by a
user-supplied subroutine. The nature of the contouring algorithm,
however, dictates that the transformation should maintain the
rectangular topology of the grid, although grid-points may be
allowed to coalesce.  As an example of a deformed rectangular
grid, consider data given on the polar grid theta=0.1n(pi/2),
for n=0,1,...,10, and r=0.25m, for m=0,1,..,4. This grid
contains 55 points, of which 11 are coincident at the origin.
The input array for PGCONX should be dimensioned (11,5), and
data values should be provided for all 55 elements.  PGCONX can
also be used for special applications in which the height of the
contour affects its appearance, e.g., stereoscopic views.
The map is truncated if necessary at the boundaries of the viewport.
Each contour line is drawn with the current line attributes (color
index, style, and width); except that if argument NC is positive
(see below), the line style is set by PGCONX to 1 (solid) for
positive contours or 2 (dashed) for negative contours. Attributes
for the contour lines can also be set in the user-supplied
subroutine, if desired.
Arguments:
 A      (input) : data array.
 IDIM   (input) : first dimension of A.
 JDIM   (input) : second dimension of A.
 I1, I2 (input) : range of first index to be contoured (inclusive).
 J1, J2 (input) : range of second index to be contoured (inclusive).
 C      (input) : array of NC contour levels; dimension at least NC.
 NC     (input) : +/- number of contour levels (less than or equal
                  to dimension of C). If NC is positive, it is the
                  number of contour levels, and the line-style is
                  chosen automatically as described above. If NC is
                  negative, it is minus the number of contour
                  levels, and the current setting of line-style is
                  used for all the contours.
 PLOT   (input) : the address (name) of a subroutine supplied by
                  the user, which will be called by PGCONX to do
                  the actual plotting. This must be declared
                  EXTERNAL in the program unit calling PGCONX.
The subroutine PLOT will be called with four arguments:
     CALL PLOT(VISBLE,X,Y,Z)
where X,Y (input) are real variables corresponding to
I,J indices of the array A. If  VISBLE (input, integer) is 1,
PLOT should draw a visible line from the current pen
position to the world coordinate point corresponding to (X,Y);
if it is 0, it should move the pen to (X,Y). Z is the value
of the current contour level, and may be used by PLOT if desired.
Example:
      SUBROUTINE PLOT (VISBLE,X,Y,Z)
      REAL X, Y, Z, XWORLD, YWORLD
      INTEGER VISBLE
      XWORLD = X*COS(Y) ! this is the user-defined
      YWORLD = X*SIN(Y) ! transformation
      IF (VISBLE.EQ.0) THEN
          CALL PGMOVE (XWORLD, YWORLD)
      ELSE
          CALL PGDRAW (XWORLD, YWORLD)
      END IF
      END
```

A.16 PGCURS – read cursor position

```
      INTEGER FUNCTION PGCURS (X, Y, CH)
      REAL X, Y
      CHARACTER*1 CH
Read the cursor position and a character typed by the user.
The position is returned in world coordinates.  PGCURS positions
the cursor at the position specified, allows the user to move the
cursor using the joystick or arrow keys or whatever is available on
the device. When he has positioned the cursor, the user types a
single character on the keyboard; PGCURS then returns this
character and the new cursor position (in world coordinates).
Returns:
 PGCURS         : 1 if the call was successful; 0 if the device
                  has no cursor or some other error occurs.
Arguments:
 X      (in/out) : the world x-coordinate of the cursor.
 Y      (in/out) : the world y-coordinate of the cursor.
 CH     (output) : the character typed by the user; if the device has
                   no cursor or if some other error occurs, the value
                   CHAR(0) [ASCII NUL character] is returned.
Note: The cursor coordinates (X,Y) may be changed by PGCURS even if
the device has no cursor or if the user does not move the cursor.
Under these circumstances, the position returned in (X,Y) is that of
the pixel nearest to the requested position.
```

A.17 PGCURSE – non-standard alias for PGCURS

```
      INTEGER FUNCTION PGCURSE (X, Y, CH)
      REAL X, Y
      CHARACTER*1 CH
```

A.18 PGDRAW – draw a line from the current pen position to a point

```
      SUBROUTINE PGDRAW (X, Y)
      REAL X, Y
Draw a line from the current pen position to the point
with world-coordinates (X,Y). The line is clipped at the edge of the
current window. The new pen position is (X,Y) in world coordinates.
Arguments:
 X      (input)  : world x-coordinate of the end point of the line.
 Y      (input)  : world y-coordinate of the end point of the line.
```

A.19 PGEBUF – end batch of output (buffer)

```
      SUBROUTINE PGEBUF
A call to PGEBUF marks the end of a batch of graphical output begun
with the last call of PGBBUF.  PGBBUF and PGEBUF calls should always
be paired. Each call to PGBBUF increments a counter, while each call
to PGEBUF decrements the counter. When the counter reaches 0, the
batch of output is written on the output device.
Arguments: none
```

A.20 PGEND – terminate PGPLOT

```
      SUBROUTINE PGEND
Terminate PGPLOT, close the plot file, release the graphics
device.  If the call to PGEND is omitted, some or all of the plot
may be lost. If the environment parameter PGPLOT_IDENT is defined
(with any value), and the device is a hardcopy device, an
identifying label is written on the plot (by calling PGIDEN: q.v.).
Arguments: none
```

A.21 PGENV – set window and viewport and draw labeled frame

```
      SUBROUTINE PGENV (XMIN, XMAX, YMIN, YMAX, JUST, AXIS)
      REAL XMIN, XMAX, YMIN, YMAX
      INTEGER JUST, AXIS
Set PGPLOT "Plotter Environment".  PGENV establishes the scaling
for subsequent calls to PGPT, PGLINE, etc.  The plotter is
advanced to a new (sub-)page, clearing the screen if necessary.
If the "prompt state" is ON (see PGASK), confirmation
is requested from the user before clearing the screen.
If requested, a box, axes, labels, etc. are drawn according to
the setting of argument AXIS.
Arguments:
 XMIN   (input)  : the world x-coordinate at the bottom left corner
                   of the viewport.
 XMAX   (input)  : the world x-coordinate at the top right corner
                   of the viewport (note XMAX may be less than XMIN).
 YMIN   (input)  : the world y-coordinate at the bottom left corner
                   of the viewport.
 YMAX   (input)  : the world y-coordinate at the top right corner
                   of the viewport (note YMAX may be less than YMIN).
 JUST   (input)  : if JUST=1, the scales of the x and y axes (in
                   world coordinates per inch) will be equal,
                   otherwise they will be scaled independently.
 AXIS   (input)  : controls the plotting of axes, tick marks, etc:
     AXIS = -2 : draw no box, axes or labels;
     AXIS = -1 : draw box only;
     AXIS =  0 : draw box and label it with coordinates;
     AXIS =  1 : same as AXIS=0, but also draw the
                 coordinate axes (X=0, Y=0);
     AXIS =  2 : same as AXIS=1, but also draw grid lines
                 at major increments of the coordinates;
     AXIS = 10 : draw box and label X-axis logarithmically;
     AXIS = 20 : draw box and label Y-axis logarithmically;
     AXIS = 30 : draw box and label both axes logarithmically.
For other axis options, use routine PGBOX. PGENV can be pursuaded to
call PGBOX with additional axis options by defining an environment
parameter PGPLOT_ENVOPT containing the required option codes.
Examples:
  PGPLOT_ENVOPT=P      ! draw Projecting tick marks
  PGPLOT_ENVOPT=I      ! Invert the tick marks
  PGPLOT_ENVOPT=IV     ! Invert tick marks and label y Vertically
```

A.22 PGERRB – horizontal error bar

```
      SUBROUTINE PGERRB (DIR, N, X, Y, E, T)
      INTEGER DIR, N
      REAL X(*), Y(*), E(*)
      REAL T
Plot error bars in the direction specified by DIR.
This routine draws an error bar only; to mark the data point at
the start of the error bar, an additional call to PGPT is required.
Arguments:
 DIR    (input)  : direction to plot the error bar relative to
                   the data point.  DIR is 1 for +X; 2 for +Y;
                   3 for -X; and 4 for -Y;
 N      (input)  : number of error bars to plot.
 X      (input)  : world x-coordinates of the data.
 Y      (input)  : world y-coordinates of the data.
 E      (input)  : value of error bar distance to be added to the
                   data position in world coordinates.
 T      (input)  : length of terminals to be drawn at the ends
                   of the error bar, as a multiple of the default
                   length; if T = 0.0, no terminals will be drawn.
Note: the dimension of arrays X, Y, and E must be greater
than or equal to N. If N is 1, X, Y, and E may be scalar
variables, or expressions.
```

A.23 PGERRX – horizontal error bar

```
      SUBROUTINE PGERRX (N, X1, X2, Y, T)
      INTEGER N
      REAL X1(*), X2(*), Y(*)
      REAL T
Plot horizontal error bars.
This routine draws an error bar only; to mark the data point in
the middle of the error bar, an additional call to PGPT or
PGERRY is required.
Arguments:
 N      (input)  : number of error bars to plot.
 X1     (input)  : world x-coordinates of lower end of the
                   error bars.
 X2     (input)  : world x-coordinates of upper end of the
                   error bars.
 Y      (input)  : world y-coordinates of the data.
 T      (input)  : length of terminals to be drawn at the ends
                   of the error bar, as a multiple of the default
                   length; if T = 0.0, no terminals will be drawn.
Note: the dimension of arrays X1, X2, and Y must be greater
than or equal to N. If N is 1, X1, X2, and Y may be scalar
variables, or expressions, eg:
      CALL PGERRX(1,X-SIGMA,X+SIGMA,Y)
```

A.24 PGERRY – vertical error bar

```
      SUBROUTINE PGERRY (N, X, Y1, Y2, T)
Plot vertical error bars.
This routine draws an error bar only; to mark the data point in
the middle of the error bar, an additional call to PGPT or
PGERRX is required.
Arguments:
 N      (input)  : number of error bars to plot.
 X      (input)  : world x-coordinates of the data.
 Y1     (input)  : world y-coordinates of top end of the
                   error bars.
 Y2     (input)  : world y-coordinates of bottom end of the
                   error bars.
 T      (input)  : length of terminals to be drawn at the ends
                   of the error bar, as a multiple of the default
                   length; if T = 0.0, no terminals will be drawn.
Note: the dimension of arrays X, Y1, and Y2 must be greater
than or equal to N. If N is 1, X, Y1, and Y2 may be scalar
variables or expressions, eg:
      CALL PGERRY(1,X,Y+SIGMA,Y-SIGMA)
```

A.25 PGETXT – erase text from graphics display

```
      SUBROUTINE PGETXT
Some graphics terminals display text (the normal interactive dialog)
on the same screen as graphics. This routine erases the text from the
view surface without affecting the graphics. It does nothing on
devices which do not display text on the graphics screen, and on
devices which do not have this capability.
Arguments:
 None
```

A.26 PGFUNT – function defined by X = F(T), Y = G(T)

```
      SUBROUTINE PGFUNT (FX, FY, N, TMIN, TMAX, PGFLAG)
      REAL FX, FY
      INTEGER N
      REAL TMIN, TMAX
      INTEGER PGFLAG
Draw a curve defined by parametric equations X = FX(T), Y = FY(T).
Arguments:
 FX     (external real function): supplied by the user, evaluates
                   X-coordinate.
 FY     (external real function): supplied by the user, evaluates
                   Y-coordinate.
 N      (input)  : the number of points required to define the
                   curve. The functions FX and FY will each be
                   called N+1 times.
 TMIN   (input)  : the minimum value for the parameter T.
 TMAX   (input)  : the maximum value for the parameter T.
 PGFLAG (input)  : if PGFLAG = 1, the curve is plotted in the
                   current window and viewport; if PGFLAG = 0,
                   PGENV is called automatically by PGFUNT to
                   start a new plot with automatic scaling.
Note: The functions FX and FY must be declared EXTERNAL in the
Fortran program unit that calls PGFUNT.
```

A.27 PGFUNX – function defined by Y = F(X)

```
      SUBROUTINE PGFUNX (FY, N, XMIN, XMAX, PGFLAG)
      REAL FY
      INTEGER N
      REAL XMIN, XMAX
      INTEGER PGFLAG
Draw a curve defined by the equation Y = FY(X), where FY is a
user-supplied subroutine.
Arguments:
 FY     (external real function): supplied by the user, evaluates
                   Y value at a given X-coordinate.
 N      (input)  : the number of points required to define the
                   curve. The function FY will be called N+1 times.
                   If PGFLAG=0 and N is greater than 1000, 1000
                   will be used instead.  If N is less than 1,
                   nothing will be drawn.
 XMIN   (input)  : the minimum value of X.
 XMAX   (input)  : the maximum value of X.
 PGFLAG (input)  : if PGFLAG = 1, the curve is plotted in the
                   current window and viewport; if PGFLAG = 0,
                   PGENV is called automatically by PGFUNX to
                   start a new plot with X limits (XMIN, XMAX)
                   and automatic scaling in Y.
Note: The function FY must be declared EXTERNAL in the Fortran
program unit that calls PGFUNX.  It has one argument, the
x-coordinate at which the y value is required, e.g.
  REAL FUNCTION FY(X)
  REAL X
  FY = .....
  END
```

A.28 PGFUNY – function defined by X = F(Y)

```
      SUBROUTINE PGFUNY (FX, N, YMIN, YMAX, PGFLAG)
      REAL    FX
      INTEGER N
      REAL    YMIN, YMAX
      INTEGER PGFLAG
Draw a curve defined by the equation X = FX(Y), where FY is a
user-supplied subroutine.
Arguments:
 FX     (external real function): supplied by the user, evaluates
                  X value at a given Y-coordinate.
 N      (input)  : the number of points required to define the
                  curve. The function FX will be called N+1 times.
                  If PGFLAG=0 and N is greater than 1000, 1000
                  will be used instead.  If N is less than 1,
                  nothing will be drawn.
 YMIN   (input)  : the minimum value of Y.
 YMAX   (input)  : the maximum value of Y.
 PGFLAG (input)  : if PGFLAG = 1, the curve is plotted in the
                  current window and viewport; if PGFLAG = 0,
                  PGENV is called automatically by PGFUNY to
                  start a new plot with Y limits (YMIN, YMAX)
                  and automatic scaling in X.
Note: The function FX must be declared EXTERNAL in the Fortran
program unit that calls PGFUNY.  It has one argument, the
y-coordinate at which the x value is required, e.g.
  REAL FUNCTION FX(Y)
  REAL Y
  FX = .....
  END
```

A.29 PGGRAY – gray-scale map of a 2D data array

```
      SUBROUTINE PGGRAY (A, IDIM, JDIM, I1, I2, J1, J2,
     1                   FG, BG, TR)
      INTEGER IDIM, JDIM, I1, I2, J1, J2
      REAL    A(IDIM,JDIM)
      REAL    FG, BG
      REAL TR(6)
```

```
Draw gray-scale map of an array in current window. The subsection
of the array A defined by indices (I1:I2, J1:J2) is mapped onto
the view surface world-coordinate system by the transformation
matrix TR. The resulting quadrilateral region is clipped at the edge
of the window and shaded with the shade at each point determined
by the corresponding array value.  The shade is a number in the
range 0 to 1 obtained by linear interpolation between the background
level (BG) and the foreground level (FG), i.e.,
  shade = [A(i,j) - BG] / [FG - BG]
The background level BG can be either less than or greater than the
foreground level FG.  Points in the array that are outside the range
BG to FG are assigned shade 0 or 1 as appropriate.
The algorithm used by PGGRAY is device-dependent.  On devices
that have only two color indices (0 and 1), the background color
is the color assigned to color index 0, the foreground color
is the color assigned to color index 1, and PGGRAY uses a
"dithering" algorithm to fill in pixels in the two colors, with
the shade (computed as above) determining the faction of pixels
that are assigned color index 1.
On devices that have more than 16 color indices, PGGRAY may use
color indices outside the range 0-15 to provide more than two
gray shades.  Note that PGGRAY may change the color representation
of these color indices, but it will not change the representation
of indices 0-15.
On most devices, the shaded region is "opaque", i.e., it obscures
all graphical elements previously drawn in the region. But on
devices that do not have erase capability, the background shade
is "transparent" and allows previously-drawn graphics to show
through.
The transformation matrix TR is used to calculate the world
coordinates of the center of the "cell" that represents each
array element. The world coordinates of the center of the cell
corresponding to array element A(I,J) are given by:
         X = TR(1) + TR(2)*I + TR(3)*J
         Y = TR(4) + TR(5)*I + TR(6)*J
Usually TR(3) and TR(5) are zero -- unless the coordinate
transformation involves a rotation or shear.  The corners of the
quadrilateral region that is shaded by PGGRAY are given by
applying this transformation to (I1-0.5,J1-0.5), (I2+0.5, J2+0.5).
Arguments:
 A      (input)  : the array to be plotted.
 IDIM   (input)  : the first dimension of array A.
 JDIM   (input)  : the second dimension of array A.
 I1, I2 (input)  : the inclusive range of the first index
                   (I) to be plotted.
 J1, J2 (input)  : the inclusive range of the second
                   index (J) to be plotted.
 FG     (input)  : the array value which is to appear with shade
                   1 ("foreground").
 BG     (input)  : the array value which is to appear with shade
                   0 ("background").
 TR     (input)  : transformation matrix between array grid and
                   world coordinates.
```

A.30 PGHI2D – cross-sections through a 2D data array

```
      SUBROUTINE PGHI2D (DATA, NXV, NYV, IX1, IX2, IY1, IY2, X, IOFF,
     1                   BIAS, CENTER, YLIMS)
      INTEGER NXV, NYV, IX1, IX2, IY1, IY2
      REAL    DATA(NXV,NYV)
      REAL    X(IX2-IX1+1), YLIMS(IX2-IX1+1)
      INTEGER IOFF
      REAL    BIAS
      LOGICAL CENTER
Plot a series of cross-sections through a 2D data array.
Each cross-section is plotted as a hidden line histogram.  The plot
can be slanted to give a pseudo-3D effect - if this is done, the
call to PGENV may have to be changed to allow for the increased X
range that will be needed.
Arguments:
 DATA   (input)  : the data array to be plotted.
 NXV    (input)  : the first dimension of DATA.
 NYV    (input)  : the second dimension of DATA.
 IX1    (input)
 IX2    (input)
 IY1    (input)
 IY2    (input)  : PGHI2D plots a subset of the input array DATA.
                   This subset is delimited in the first (x)
                   dimension by IX1 and IX2 and the 2nd (y) by IY1
                   and IY2, inclusively. Note: IY2 < IY1 is
                   permitted, resulting in a plot with the
                   cross-sections plotted in reverse Y order.
                   However, IX2 must be => IX1.
 X      (input)  : the abscissae of the bins to be plotted. That is,
                   X(1) should be the X value for DATA(IX1,IY1), and
                   X should have (IX2-IX1+1) elements.  The program
                   has to assume that the X value for DATA(x,y) is
                   the same for all y.
 IOFF   (input)  : an offset in array elements applied to successive
                   cross-sections to produce a slanted effect.  A
                   plot with IOFF > 0 slants to the right, one with
                   IOFF < 0 slants left.
 BIAS   (input)  : a bias value applied to each successive cross-
                   section in order to raise it above the previous
                   cross-section.  This is in the same units as the
                   data.
 CENTER (input)  : if .true., the X values denote the center of the
                   bins; if .false. the X values denote the lower
                   edges (in X) of the bins.
 YLIMS  (input)  : workspace.  Should be an array of at least
                   (IX2-IX1+1) elements.
```

A.31 PGHIST – histogram of unbinned data

```
      SUBROUTINE PGHIST (N, DATA, DATMIN, DATMAX, NBIN, PGFLAG)
      INTEGER N
      REAL    DATA(*)
      REAL    DATMIN, DATMAX
      INTEGER NBIN, PGFLAG
Draw a histogram of N values of a variable in array
DATA(1...N) in the range DATMIN to DATMAX using NBIN bins.  Note
that array elements which fall exactly on the boundary between
two bins will be counted in the higher bin rather than the
lower one; and array elements whose value is less than DATMIN or
greater than or equal to DATMAX will not be counted at all.
Arguments:
 N      (input)  : the number of data values.
 DATA   (input)  : the data values. Note: the dimension of array
                   DATA must be greater than or equal to N. The
                   first N elements of the array are used.
 DATMIN (input)  : the minimum data value for the histogram.
 DATMAX (input)  : the maximum data value for the histogram.
 NBIN   (input)  : the number of bins to use: the range DATMIN to
                   DATMAX is divided into NBIN equal bins and
                   the number of DATA values in each bin is
                   determined by PGHIST.  NBIN may not exceed 200.
 PGFLAG (input)  : if PGFLAG = 1, the histogram is plotted in the
                   current window and viewport; if PGFLAG = 0,
                   PGENV is called automatically by PGHIST to start
                   a new plot (the x-limits of the window will be
                   DATMIN and DATMAX; the y-limits will be chosen
                   automatically.
```

A.32 PGIDEN – write username, date, and time at bottom of plot

```
      SUBROUTINE PGIDEN
Write username, date, and time at bottom of plot.
Arguments: none.
```

A.33 PGLAB – write labels for x-axis, y-axis, and top of plot

```
      SUBROUTINE PGLAB (XLBL, YLBL, TOPLBL)
      CHARACTER*(*) XLBL, YLBL, TOPLBL
Write labels outside the viewport. This routine is a simple
interface to PGMTXT, which should be used if PGLAB is inadequate.
Arguments:
 XLBL   (input) : a label for the x-axis (centered below the
                  viewport).
 YLBL   (input) : a label for the y-axis (centered to the left
                  of the viewport, drawn vertically).
 TOPLBL (input) : a label for the entire plot (centered above the
                  viewport).
```

A.34 PGLABEL – non-standard alias for PGLAB

```
      SUBROUTINE PGLABEL (XLBL, YLBL, TOPLBL)
      CHARACTER*(*) XLBL, YLBL, TOPLBL
```

A.35 PGLCUR – draw a line using the cursor

```
      SUBROUTINE PGLCUR (MAXPT, NPT, X, Y)
      INTEGER MAXPT, NPT
      REAL    X(*), Y(*)
Interactive routine for user to enter a polyline by use of
the cursor.  Routine allows user to Add and Delete vertices;
vertices are joined by straight-line segments.
Arguments:
 MAXPT  (input)  : maximum number of points that may be accepted.
 NPT    (in/out) : number of points entered; should be zero on
                   first call.
 X      (in/out) : array of x-coordinates (dimension at least MAXPT).
 Y      (in/out) : array of y-coordinates (dimension at least MAXPT).
Notes:
(1) On return from the program, cursor points are returned in
the order they were entered. Routine may be (re-)called with points
already defined in X,Y (# in NPT), and they will be plotted
first, before editing.
(2) User commands: the user types single-character commands
after positioning the cursor: the following are accepted:
  A (Add)    - add point at current cursor location.
  D (Delete) - delete last-entered point.
  X (eXit)   - leave subroutine.
```

A.36 PGLDEV – list available device types

```
      SUBROUTINE PGLDEV
Writes a list to the terminal of all device types known to the
current version of PGPLOT.
Arguments: none.
```

A.37 PGLEN – Find length of a string in a variety of units

```
      SUBROUTINE PGLEN (UNITS, STRING, XL, YL)
      REAL XL, YL
      INTEGER UNITS
      CHARACTER*(*) STRING
Work out length of a string in x and y directions
Input
 UNITS    :  0 => answer in normalized device coordinates
             1 => answer in inches
             2 => answer in mm
             3 => answer in absolute device coordinates (dots)
             4 => answer in world coordinates
             5 => answer as a fraction of the current viewport size
 STRING   :  String of interest
Output
 XL       :  Length of string in x direction
 YL       :  Length of string in y direction
```

A.38 PGLINE – draw a polyline (curve defined by line-segments)

```
      SUBROUTINE PGLINE (N, XPTS, YPTS)
      INTEGER  N
      REAL     XPTS(*), YPTS(*)
Primitive routine to draw a Polyline. A polyline is one or more
connected straight-line segments.  The polyline is drawn using
the current setting of attributes color-index, line-style, and
line-width. The polyline is clipped at the edge of the window.
Arguments:
 N      (input)  : number of points defining the line; the line
                   consists of (N-1) straight-line segments.
                   N should be greater than 1 (if it is 1 or less,
                   nothing will be drawn).
 XPTS   (input)  : world x-coordinates of the points.
 YPTS   (input)  : world y-coordinates of the points.
The dimension of arrays X and Y must be greater than or equal to N.
The "pen position" is changed to (X(N),Y(N)) in world coordinates
(if N > 1).
```

A.39 PGMOVE – move pen (change current pen position)

```
      SUBROUTINE PGMOVE (X, Y)
      REAL X, Y
Primitive routine to move the "pen" to the point with world
coordinates (X,Y). No line is drawn.
Arguments:
 X      (input)  : world x-coordinate of the new pen position.
 Y      (input)  : world y-coordinate of the new pen position.
```

A.40 PGMTEXT – non-standard alias for PGMTXT

```
      SUBROUTINE PGMTEXT (SIDE, DISP, COORD, FJUST, TEXT)
      CHARACTER*(*) SIDE, TEXT
      REAL DISP, COORD, FJUST
```

A.41 PGMTXT – write text at position relative to viewport

```
      SUBROUTINE PGMTXT (SIDE, DISP, COORD, FJUST, TEXT)
      CHARACTER*(*) SIDE, TEXT
      REAL DISP, COORD, FJUST
Write text at a position specified relative to the viewport (outside
or inside).  This routine is useful for annotating graphs. It is used
by routine PGLAB.  The text is written using the current values of
attributes color-index, line-width, character-height, and
character-font.
Arguments:
 SIDE   (input)  : must include one of the characters 'B', 'L', 'T',
                   or 'R' signifying the Bottom, Left, Top, or Right
                   margin of the viewport. If it includes 'LV' or
                   'RV', the string is written perpendicular to the
                   frame rather than parallel to it.
 DISP   (input)  : the displacement of the character string from the
                   specified edge of the viewport, measured outwards
                   from the viewport in units of the character
                   height. Use a negative value to write inside the
                   viewport, a positive value to write outside.
 COORD  (input)  : the location of the character string along the
                   specified edge of the viewport, as a fraction of
                   the length of the edge.
 FJUST  (input)  : controls justification of the string parallel to
                   the specified edge of the viewport. If
                   FJUST = 0.0, the left-hand end of the string will
                   be placed at COORD; if JUST = 0.5, the center of
                   the string will be placed at COORD; if JUST = 1.0,
                   the right-hand end of the string will be placed at
                   at COORD. Other values between 0 and 1 give inter-
                   mediate placing, but they are not very useful.
 TEXT   (input) :  the text string to be plotted. Trailing spaces are
                   ignored when justifying the string, but leading
                   spaces are significant.
```

A.42 PGNCUR – mark a set of points using the cursor

```
      SUBROUTINE PGNCUR (MAXPT, NPT, X, Y, SYMBOL)
      INTEGER MAXPT, NPT
      REAL    X(*), Y(*)
      INTEGER SYMBOL
Interactive routine for user to enter data points by use of
the cursor.  Routine allows user to Add and Delete points.  The
points are returned in order of increasing x-ccordinate, not in the
order they were entered.
Arguments:
 MAXPT  (input)  : maximum number of points that may be accepted.
 NPT    (in/out) : number of points entered; should be zero on
                   first call.
 X      (in/out) : array of x-coordinates.
 Y      (in/out) : array of y-coordinates.
 SYMBOL (input)  : code number of symbol to use for marking
                   entered points (see PGPT).
Note (1): The dimension of arrays X and Y must be greater than or
equal to MAXPT.
Note (2): On return from the program, cursor points are returned in
increasing order of X. Routine may be (re-)called with points
already defined in X,Y (number in NPT), and they will be plotted
first, before editing.
Note (3): User commands: the user types single-character commands
after positioning the cursor: the following are accepted:
A (Add)    - add point at current cursor location.
D (Delete) - delete nearest point to cursor.
X (eXit)   - leave subroutine.
```

A.43 PGNCURSE – non-standard alias for PGNCUR

```
      SUBROUTINE PGNCURSE (MAXPT, NPT, X, Y, SYMBOL)
      INTEGER MAXPT, NPT
      REAL    X(*), Y(*)
      INTEGER SYMBOL
```

A.44 PGNUMB – convert a number into a plottable character string

```
      SUBROUTINE PGNUMB (MM, PP, FORM, STRING, NC)
      INTEGER MM, PP, FORM
      CHARACTER*(*) STRING
      INTEGER NC
This routine converts a number into a decimal character
representation. To avoid problems of floating-point roundoff, the
number must be provided as an integer (MM) multiplied by a power of 10
(10**PP).  The output string retains only significant digits of MM,
and will be in either integer format (123), decimal format (0.0123),
or exponential format (1.23x10**5). Standard escape sequences \u, \d
raise the exponent and \\ is used for the multiplication sign.
This routine is used by PGBOX to create numeric labels for a plot.
Formatting rules:
  (a) Decimal notation (FORM=1):
      - Trailing zeros to the right of the decimal sign are
        omitted
      - The decimal sign is omitted if there are no digits
        to the right of it
      - When the decimal sign is placed before the first digit
        of the number, a zero is placed before the decimal sign
      - The decimal sign is a period (.)
      - No spaces are placed between digits (ie digits are not
        grouped in threes as they should be)
      - A leading minus (-) is added if the number is negative
  (b) Exponential notation (FORM=2):
      - The exponent is adjusted to put just one (non-zero)
        digit before the decimal sign
      - The mantissa is formatted as in (a), unless its value is
        1 in which case it and the multiplication sign are omitted
      - If the power of 10 is not zero and the mantissa is not
        zero, an exponent of the form \\10\u[-]nnn is appended,
        where \\ is a multiplication sign (cross), \u is an escape
        sequence to raise the exponent, and as many digits nnn
        are used as needed
  (c) Automatic choice (FORM=0):
        Decimal notation is used if the absolute value of the
        number is less than 10000 or greater than or equal to
        0.01. Otherwise exponential notation is used.
Arguments:
 MM     (input)
 PP     (input)  : the value to be formatted is MM*10**PP.
 FORM   (input)  : controls how the number is formatted:
                   FORM = 0 -- use either decimal or exponential
                   FORM = 1 -- use decimal notation
                   FORM = 2 -- use exponential notation
 STRING (output) : the formatted character string, left justified.
                   If the length of STRING is insufficient, a single
                   asterisk is returned, and NC=1.
 NC     (output) : the number of characters used in STRING:
                   the string to be printed is STRING(1:NC).
```

A.45 PGOLIN – mark a set of points using the cursor

```
      SUBROUTINE PGOLIN (MAXPT, NPT, X, Y, SYMBOL)
      INTEGER MAXPT, NPT
      REAL    X(*), Y(*)
      INTEGER SYMBOL
Interactive routine for user to enter data points by use of
the cursor.  Routine allows user to Add and Delete points.  The
points are returned in the order that they were entered (unlike
PGNCUR).
Arguments:
 MAXPT  (input)  : maximum number of points that may be accepted.
 NPT    (in/out) : number of points entered; should be zero on
                   first call.
 X      (in/out) : array of x-coordinates.
 Y      (in/out) : array of y-coordinates.
 SYMBOL (input)  : code number of symbol to use for marking
                   entered points (see PGPT).
Note (1): The dimension of arrays X and Y must be greater than or
equal to MAXPT.
Note (2): On return from the program, cursor points are returned in
the order they were entered. Routine may be (re-)called with points
already defined in X,Y (number in NPT), and they will be plotted
first, before editing.
Note (3): User commands: the user types single-character commands
after positioning the cursor: the following are accepted:
A (Add)    - add point at current cursor location.
D (Delete) - delete the last point entered.
X (eXit)   - leave subroutine.
```

A.46 PGPAGE – advance to new page

```
      SUBROUTINE PGPAGE
Advance plotter to a new (sub-)page, clearing the screen if
necessary. If the "prompt state" is ON (see PGASK), confirmation is
requested from the user before clearing the screen.  For an
explanation of sub-pages, see PGBEG.  PGPAGE does not change the
window or the position of the viewport relative to the (sub-)page.
Arguments: none
```

A.47 PGPAP – change the size of the view surface ("paper size")

```
      SUBROUTINE PGPAP (WIDTH, ASPECT)
      REAL WIDTH, ASPECT
This routine changes the size of the view surface to a specified
width and aspect ratio (height/width), in so far as this is possible
on the specific device. It is always possible to obtain a view
surface smaller than the default size; on some devices (e.g.,
printers that print on roll or fan-feed paper) it is possible
to obtain a view surface larger than the default. If this routine is
used, it must be called immediately after PGBEGIN.
Arguments:
 WIDTH  (input)  : the requested width of the view surface in inches;
                   if WIDTH=0.0, PGPAP will obtain the largest view
                   surface available consistent with argument ASPECT.
 ASPECT (input)  : the aspect ratio (height/width) of the view
                   surface; e.g., ASPECT=1.0 gives a square view
                   surface, ASPECT=0.618 gives a horizontal
                   rectangle, ASPECT=1.618 gives a vertical rectangle.
```

A.48 PGPAPER – non-standard alias for PGPAP

```
      SUBROUTINE PGPAPER (WIDTH, ASPECT)
      REAL WIDTH, ASPECT
```

A.49 PGPIXL – draw pixels

```
      SUBROUTINE PGPIXL (IA, IDIM, JDIM, I1, I2, J1, J2,
     1                   X1, X2, Y1, Y2)
      INTEGER IDIM, JDIM, I1, I2, J1, J2
      INTEGER IA(IDIM,JDIM)
      REAL    X1, X2, Y1, Y2
Draw lots of solid-filled (tiny) rectangles alligned with the
coordinate axes. Best performance is achieved when output is
directed to a pixel-oriented device and the rectangles coincide
with the pixels on the device. In other cases, pixel output is
emulated.
The subsection of the array IA defined by indices (I1:I2, J1:J2)
is mapped onto world-coordinate rectangle defined by X1, X2, Y1
and Y2. This rectangle is divided int (I2 - I1 + 1) * (J2 - J1 + 1)
small rectangles. Each of these small rectangles is solid-filled
with the color index specified by the corresponding element of
IA.
On most devices, the output region is "opaque", i.e., it obscures
all graphical elements previously drawn in the region. But on
devices that do not have erase capability, the background shade
is "transparent" and allows previously-drawn graphics to show
through.
Arguments:
 IA     (input)  : the array to be plotted.
 IDIM   (input)  : the first dimension of array A.
 JDIM   (input)  : the second dimension of array A.
 I1, I2 (input)  : the inclusive range of the first index
                   (I) to be plotted.
 J1, J2 (input)  : the inclusive range of the second
                   index (J) to be plotted.
 X1, Y1 (input)  : world coordinates of one corner of the output
                   region
 X2, Y2 (input)  : world coordinates of the opposite corner of the
                   output region
```

A.50 PGPNTS – draw one or more graph markers

```
      SUBROUTINE PGPNTS (N, X, Y, SYMBOL, NS)
      INTEGER N, NS
      REAL X(*), Y(*)
      INTEGER SYMBOL(*)
Draw Graph Markers. Unlike PGPT, this routine can draw a different
symbol at each point. The markers
are drawn using the current values of attributes color-index,
line-width, and character-height (character-font applies if the symbol
number is >31).  If the point to be marked lies outside the window,
no marker is drawn.  The "pen position" is changed to
(XPTS(N),YPTS(N)) in world coordinates (if N > 0).
Arguments:
 N      (input)  : number of points to mark.
 X      (input)  : world x-coordinate of the points.
 Y      (input)  : world y-coordinate of the points.
 SYMBOL (input)  : code number of the symbol to be plotted at each
                   point (see PGPT).
 NS     (input)  : number of values in the SYMBOL array.  If NS <= N,
                   then the first NS points are drawn using the value
                   of SYMBOL(I) at (X(I), Y(I)) and SYMBOL(1) for all
                   the values of (X(I), Y(I)) where I > NS.
Note: the dimension of arrays X and Y must be greater than or equal
to N and the dimension of the array SYMBOL must be greater than or
equal to NS.  If N is 1, X and Y may be scalars (constants or
variables).  If NS is 1, then SYMBOL may be a scalar.  If N is
less than 1, nothing is drawn.
```

A.51 PGPOINT – non-standard alias for PGPT

```
      SUBROUTINE PGPOINT (N, XPTS, YPTS, SYMBOL)
      INTEGER N
      REAL XPTS(*), YPTS(*)
      INTEGER SYMBOL
```

A.52 PGPOLY – fill a polygonal area with shading

```
      SUBROUTINE PGPOLY (N, XPTS, YPTS)
      INTEGER N
      REAL XPTS(*), YPTS(*)
Fill-area primitive routine: shade the interior of a closed
polygon in the current window.  The action of this routine depends
on the setting of the Fill-Area Style attribute. If Fill-Area Style
is SOLID (the default), the interior of the polygon is solid-filled
using the current Color Index. If Fill-Area Style is HOLLOW, the
outline of the polygon is drwan using the current line attributes
(color index, line-style, and line-width). Other values of the Fill-
Area attribute may be allowed in future, e.g., for shading with
patterns or hatching. The polygon is clipped at the edge of the
window. The pen position is changed to (XPTS(1),YPTS(1)) in world
coordinates (if N > 1).  If the polygon is not convex, a point is
assumed to lie inside the polygon if a straight line drawn to
infinity intersects and odd number of the polygon's edges.
Arguments:
 N      (input)  : number of points defining the polygon; the
                   line consists of N straight-line segments,
                   joining points 1 to 2, 2 to 3,... N-1 to N, N to 1.
                   N should be greater than 2 (if it is 2 or less,
                   nothing will be drawn).
 XPTS   (input)  : world x-coordinates of the vertices.
 YPTS   (input)  : world y-coordinates of the vertices.
                   Note: the dimension of arrays XPTS and YPTS must be
                   greater than or equal to N.
```

A.53 PGPT – draw one or more graph markers

```
      SUBROUTINE PGPT (N, XPTS, YPTS, SYMBOL)
      INTEGER N
      REAL XPTS(*), YPTS(*)
      INTEGER SYMBOL
Primitive routine to draw Graph Markers (polymarker). The markers
are drawn using the current values of attributes color-index,
line-width, and character-height (character-font applies if the symbol
number is >31).  If the point to be marked lies outside the window,
no marker is drawn.  The "pen position" is changed to
(XPTS(N),YPTS(N)) in world coordinates (if N > 0).
Arguments:
 N      (input)  : number of points to mark.
 XPTS   (input)  : world x-coordinates of the points.
 YPTS   (input)  : world y-coordinates of the points.
 SYMBOL (input)  : code number of the symbol to be drawn at each
                   point:
                   -1, -2  : a single dot (diameter = current
                             line width).
                   -3..-31 : a regular polygon with ABS(SYMBOL)
                             edges (style set by current fill style).
                   0..31   : standard marker symbols.
                   32..127 : ASCII characters (in current font).
                             e.g. to use letter F as a marker, let
                             SYMBOL = ICHAR('F').
                   > 127  :  a Hershey symbol number.
Note: the dimension of arrays X and Y must be greater than or equal
to N. If N is 1, X and Y may be scalars (constants or variables). If
N is less than 1, nothing is drawn.
```

A.54 PGPTEXT – non-standard alias for PGPTXT

```
      SUBROUTINE PGPTEXT (X, Y, ANGLE, FJUST, TEXT)
      REAL X, Y, ANGLE, FJUST
      CHARACTER*(*) TEXT
```

A.55 PGPTXT – write text at arbitrary position and angle

```
      SUBROUTINE PGPTXT (X, Y, ANGLE, FJUST, TEXT)
      REAL X, Y, ANGLE, FJUST
      CHARACTER*(*) TEXT
Primitive routine for drawing text. The text may be drawn at any
angle with the horizontal, and may be centered or left- or right-
justified at a specified position.  Routine PGTEXT provides a
simple interface to PGPTXT for horizontal strings. Text is drawn
using the current values of attributes color-index, line-width,
character-height, and character-font.  Text is NOT subject to
clipping at the edge of the window.
Arguments:
 X      (input)  : world x-coordinate.
 Y      (input)  : world y-coordinate. The string is drawn with the
                   baseline of all the characters passing through
                   point (X,Y); the positioning of the string along
                   this line is controlled by argument FJUST.
 ANGLE  (input)  : angle, in degrees, that the baseline is to make
                   with the horizontal, increasing counter-clockwise
                   (0.0 is horizontal).
 FJUST  (input)  : controls horizontal justification of the string.
                   If FJUST = 0.0, the string will be left-justified
                   at the point (X,Y); if FJUST = 0.5, it will be
                   centered, and if FJUST = 1.0, it will be right
                   justified. [Other values of FJUST give other
                   justifications.]
 TEXT   (input)  : the character string to be plotted.
```

A.56 PGQCF – inquire character font

```
      SUBROUTINE PGQCF (IF)
      INTEGER  IF
Query the current Character Font (set by routine PGSCF).
Argument:
 IF     (output)   : the current font number (in range 1-4).
```

A.57 PGQCH – inquire character height

```
      SUBROUTINE PGQCH (SIZE)
      REAL SIZE
Query the Character Size attribute (set by routine PGSCH).
Argument:
 SIZE   (output) : current character size (dimensionless multiple of
                   the default size).
```

A.58 PGQCI – inquire color index

```
      SUBROUTINE PGQCI (CI)
      INTEGER  CI
Query the Color Index attribute (set by routine PGSCI).
Argument:
 CI     (output) : the current color index (in range 0-max). This is
                   the color index actually in use, and may differ
                   from the color index last requested by PGSCI if
                   that index is not available on the output device.
```

A.59 PGQCOL – inquire color capability

```
      SUBROUTINE PGQCOL (CI1, CI2)
      INTEGER  CI1, CI2
Query the range of color indices available on the current device.
Argument:
 CI1    (output) : the minimum available color index. This will be
                   either 0 if the device can write in the
                   background color, or 1 if not.
 CI2    (output) : the maximum available color index. This will be
                   1 if the device has no color capability, or a
                   larger number (e.g., 3, 7, 15, 255).
```

A.60 PGQCR – inquire color representation

```
PGQCR: not yet available.
```

A.61 PGQFS – inquire fill-area style

```
      SUBROUTINE PGQFS (FS)
      INTEGER  FS
Query the current Fill-Area Style attribute (set by routine
PGSFS).
Argument:
 FS     (output) : the current fill-area style:
                     FS = 1 => solid (default)
                     FS = 2 => hollow
```

A.62 PGQINF – inquire PGPLOT general information

```
      SUBROUTINE PGQINF (ITEM, VALUE, LENGTH)
      CHARACTER*(*) ITEM, VALUE
      INTEGER LENGTH
This routine can be used to obtain miscellaneous information about
the PGPLOT environment. Input is a character string defining the
information required, and output is a character string containing the
requested information.
The following item codes are accepted (note that the strings must
match exactly, except for case, but only the first 8 characters are
significant). For items marked *, PGPLOT must be in the OPEN state
for the inquiry to succeed. If the inquiry is unsuccessful, either
because the item code is not recognized or because the information
is not available, a question mark ('?') is returned.
  'VERSION'     - version of PGPLOT software in use.
  'STATE'       - status of PGPLOT ('OPEN' if a graphics device
                  is open for output, 'CLOSED' otherwise).
  'USER'        - the username associated with the calling program.
  'NOW'         - current date and time (e.g., '17-FEB-1986 10:04').
  'DEVICE'    * - current PGPLOT device or file.
  'FILE'      * - current PGPLOT device or file.
  'TYPE'      * - device-type of the current PGPLOT device.
  'DEV/TYPE'  * - current PGPLOT device and type, in a form which
                  is acceptable as an argument for PGBEG.
  'HARDCOPY'  * - is the current device a hardcopy device? ('YES' or
                  'NO').
  'TERMINAL'  * - is the current device the user's interactive
                  terminal? ('YES' or 'NO').
  'CURSOR'    * - does the current device have a graphics cursor?
                  ('YES' or 'NO').
Arguments:
 ITEM  (input)  : character string defining the information to
                  be returned; see above for a list of possible
                  values.
 VALUE (output) : returns a character-string containing the
                  requested information.
 LENGTH (output): the number of characters returned in VALUE
                  (VALUE is padded with spaces to the length
                  supplied).
```

A.63 PGQLS – inquire line style

```
      SUBROUTINE PGQLS (LS)
      INTEGER  LS
Query the current Line Style attribute (set by routine PGSLS).
Argument:
 LS     (output) : the current line-style attribute (in range 1-5).
```

A.64 PGQLW – inquire line width

```
      SUBROUTINE PGQLW (LW)
      INTEGER  LW
Query the current Line-Width attribute (set by routine PGSLW).
Argument:
 LW     (output)  : the line-width (in range 1-21).
```

A.65 PGQPOS – inquire current pen position

```
      SUBROUTINE PGQPOS (X, Y)
      REAL X, Y
Query the current "pen" position in world C coordinates (X,Y).
Arguments:
 X      (output)  : world x-coordinate of the pen position.
 Y      (output)  : world y-coordinate of the pen position.
```

A.66 PGQVP – inquire viewport size and position

```
      SUBROUTINE PGQVP (UNITS, X1, X2, Y1, Y2)
      INTEGER UNITS
      REAL    X1, X2, Y1, Y2
Inquiry routine to determine the current viewport setting.
The values returned may be normalized device coordinates, inches, mm,
or pixels, depending on the value of the input parameter CFLAG.
Arguments:
 UNITS  (input)  : used to specify the units of the output parameters:
                   UNITS = 0 : normalized device coordinates
                   UNITS = 1 : inches
                   UNITS = 2 : millimeters
                   UNITS = 3 : pixels
                   Other values give an error message, and are
                   treated as 0.
 X1     (output) : the x-coordinate of the bottom left corner of the
                   viewport.
 X2     (output) : the x-coordinate of the top right corner of the
                   viewport.
 Y1     (output) : the y-coordinate of the bottom left corner of the
                   viewport.
 Y2     (output) : the y-coordinate of the top right corner of the
                   viewport.
```

A.67 PGQWIN – inquire window boundary coordinates

```
      SUBROUTINE PGQWIN (X1, X2, Y1, Y2)
      REAL X1, X2, Y1, Y2
Inquiry routine to determine the current window setting.
The values returned are world coordinates.
Arguments:
 X1     (output) : the x-coordinate of the bottom left corner
                   of the window.
 X2     (output) : the x-coordinate of the top right corner
                   of the window.
 Y1     (output) : the y-coordinate of the bottom left corner
                   of the window.
 Y2     (output) : the y-coordinate of the top right corner
                   of the window.
```

A.68 PGRECT – draw a rectangle, using fill-area attributes

```
      SUBROUTINE PGRECT (X1, X2, Y1, Y2)
      REAL X1, X2, Y1, Y2
This routine can be used instead of PGPOLY for the special case of
drawing a rectangle aligned with the coordinate axes; only two
vertices need be specified instead of four.  On most devices, it is
faster to use PGRECT than PGPOLY for drawing rectangles.  The
rectangle has vertices at (X1,Y1), (X1,Y2), (X2,Y2), and (X1,Y2).
Arguments:
 X1, X2 (input) : the horizontal range of the rectangle.
 Y1, Y2 (input) : the vertical range of the rectangle.
```

A.69 PGRND – find the smallest "round" number greater than x

```
      REAL FUNCTION PGRND (X, NSUB)
      REAL X
      INTEGER NSUB
Rooutine to find the smallest "round" number larger than x, a
"round" number being 1, 2 or 5 times a power of 10. If X is negative,
PGRND(X) = -PGRND(ABS(X)). eg PGRND(8.7) = 10.0,
PGRND(-0.4) = -0.5.  If X is zero, the value returned is zero.
CThis routine is used by PGBOX for choosing  tick intervals.
Returns:
 PGRND         : the "round" number.
Arguments:
 X      (input)  : the number to be rounded.
 NSUB   (output) : a suitable number of subdivisions for
                   subdividing the "nice" number: 2 or 5.
```

A.70 PGRNGE – choose axis limits

```
      SUBROUTINE PGRNGE (X1, X2, XLO, XHI)
      REAL X1, X2, XLO, XHI
Choose plotting limits XLO and XHI which encompass the data
range X1 to X2.
Arguments:
 X1, X2 (input)  : the data range (X1<X2), ie, the min and max values
                   to be plotted.
 XLO
 XHI    (output) : suitable values to use as the extremes of a graph
                   axis (XLO <= X1, XHI >= X2).
```

A.71 PGSCF – set character font

```
      SUBROUTINE PGSCF (IF)
      INTEGER  IF
Set the Character Font for subsequent text plotting. Four different
fonts are available:
  1: (default) a simple single-stroke font ("normal" font)
  2: roman font
  3: italic font
  4: script font
This call determines which font is in effect at the beginning of
each text string. The font can be changed (temporarily) within a text
string by using the escape sequences \fn, \fr, \fi, and \fs for fonts
1, 2, 3, and 4, respectively.
Argument:
 IF     (input)  : the font number to be used for subsequent text
                   plotting (in range 1-4).
```

A.72 PGSCH – set character height

```
      SUBROUTINE PGSCH (SIZE)
      REAL SIZE
Set the character size attribute. The size affects all text and graph
markers drawn later in the program. The default character size is
1.0, corresponding to a character height about 1/40 the height of
the view surface.  Changing the character size also scales the length
of tick marks drawn by PGBOX and terminals drawn by PGERRX and PGERRY.
Argument:
 SIZE   (input)  : new character size (dimensionless multiple of
                   the default size).
```

A.73 PGSCI – set color index

```
      SUBROUTINE PGSCI (CI)
      INTEGER  CI
Set the Color Index for subsequent plotting, if the output device
permits this. The default color index is 1, usually white on a black
background for video displays or black on a white background for
printer plots. The color index is an integer in the range 0 to a
device-dependent maximum. Color index 0 corresponds to the background
color; lines may be "erased" by overwriting them with color index 0
(if the device permits this).
If the requested color index is not available on the selected device,
color index 1 will be substituted.
The assignment of colors to color indices can be changed with
subroutine PGSCR (set color representation).  Color indices 0-15
have predefined color representations (see the PGPLOT manual), but
these may be changed with PGSCR.  Color indices above 15  have no
predefined representations: if these indices are used, PGSCR must
be called to define the representation.
Argument:
 CI     (input)  : the color index to be used for subsequent plotting
                   on the current device (in range 0-max). If the
                   index exceeds the device-dependent maximum, the
                   default color index (1) is used.
```

A.74 PGSCR – set color representation

```
      SUBROUTINE PGSCR (CI, CR, CG, CB)
      INTEGER CI
      REAL    CR, CG, CB
Set color representation: i.e., define the color to be
associated with a color index.  Ignored for devices which do not
support variable color or intensity.  Color indices 0-15
have predefined color representations (see the PGPLOT manual), but
these may be changed with PGSCR.  Color indices 16-maximum have no
predefined representations: if these indices are used, PGSCR must
be called to define the representation. On monochrome output
devices (e.g. VT125 terminals with monochrome monitors), the
monochrome intensity is computed from the specified Red, Green, Blue
intensities as 0.30*R + 0.59*G + 0.11*B, as in US color television
systems, NTSC encoding.  Note that most devices do not have an
infinite range of colors or monochrome intensities available;
the nearest available color is used.  Examples: for black,
set CR=CG=CB=0.0; for white, set CR=CG=CB=1.0; for medium gray,
set CR=CG=CB=0.5; for medium yellow, set CR=CG=0.5, CB=0.0.
Argument:
 CI     (input)  : the color index to be defined, in the range 0-max.
                   If the color index greater than the device
                   maximum is specified, the call is ignored. Color
                   index 0 applies to the background color.
 CR     (input)  : red, green, and blue intensities,
 CG     (input)    in range 0.0 to 1.0.
 CB     (input)
```

A.75 PGSFS – set fill-area style

```
      SUBROUTINE PGSFS (FS)
      INTEGER  FS
Set the Fill-Area Style attribute for subsequent area-fill by
PGPOLY.  At present only two styles are available: solid (fill
polygon with solid color of the current color-index), and hollow
(draw outline of polygon only, using current line attributes).
Argument:
 FS     (input)  : the fill-area style to be used for subsequent
                   plotting:
                     FS = 1 => solid (default)
                     FS = 2 => hollow
                   Other values give an error message and are
                   treated as 2.
```

A.76 PGSHLS – set color representation using HLS system

```
      SUBROUTINE PGSHLS (CI, CH, CL, CS)
      INTEGER CI
      REAL    CH, CL, CS
Set color representation: i.e., define the color to be
associated with a color index.  This routine is equivalent to
PGSCR, but the color is defined in the Hue-Lightness-Saturation
model instead of the Red-Green-Blue model.
Reference: SIGGRAPH Status Report of the Graphic Standards Planning
Committee, Computer Graphics, Vol.13, No.3, Association for
Computing Machinery, New York, NY, 1979.
Argument:
 CI     (input)  : the color index to be defined, in the range 0-max.
                   If the color index greater than the device
                   maximum is specified, the call is ignored. Color
                   index 0 applies to the background color.
 CH     (input)  : hue, in range 0.0 to 360.0.
 CL     (input)  : lightness, in range 0.0 to 1.0.
 CS     (input)  : saturation, in range 0.0 to 1.0.
```

A.77 PGSLS – set line style

```
      SUBROUTINE PGSLS (LS)
      INTEGER  LS
Set the line style attribute for subsequent plotting. This
attribute affects line primitives only; it does not affect graph
markers, text, or area fill.
Five different line styles are available, with the following codes:
1 (full line), 2 (dashed), 3 (dot-dash-dot-dash), 4 (dotted),
5 (dash-dot-dot-dot). The default is 1 (normal full line).
Argument:
 LS     (input)  : the line-style code for subsequent plotting
                   (in range 1-5).
```

A.78 PGSLW – set line width

```
      SUBROUTINE PGSLW (LW)
      INTEGER  LW
Set the line-width attribute. This attribute affects lines, graph
markers, and text. Thick lines are generated by tracing each line
with multiple strokes offset in the direction perpendicular to the
line. The line width is specified by the number of strokes to be
used, which must be in the range 1-201. The actual line width
obtained depends on the device resolution.
Argument:
 LW     (input)  : the number of strokes to be used
                   (in range 1-201).
```

A.79 PGSVP – set viewport (normalized device coordinates)

```
      SUBROUTINE PGSVP (XLEFT, XRIGHT, YBOT, YTOP)
      REAL XLEFT, XRIGHT, YBOT, YTOP
Change the size and position of the viewport, specifying
the viewport in normalized device coordinates.  Normalized
device coordinates run from 0 to 1 in each dimension. The
viewport is the rectangle on the view surface "through"
which one views the graph.  All the PG routines which plot lines
etc. plot them within the viewport, and lines are truncated at
the edge of the viewport (except for axes, labels etc drawn with
PGBOX or PGLAB).  The region of world space (the coordinate
space of the graph) which is visible through the viewport is
specified by a call to PGSWIN.  It is legal to request a
viewport larger than the view surface; only the part which
appears on the view surface will be plotted.
Arguments:
 XLEFT  (input)  : x-coordinate of left hand edge of viewport, in NDC.
 XRIGHT (input)  : x-coordinate of right hand edge of viewport,
                   in NDC.
 YBOT   (input)  : y-coordinate of bottom edge of viewport, in NDC.
 YTOP   (input)  : y-coordinate of top  edge of viewport, in NDC.
```

A.80 PGSWIN – set window

```
      SUBROUTINE PGSWIN (X1, X2, Y1, Y2)
      REAL X1, X2, Y1, Y2
Change the window in world coordinate space that is to be mapped on
to the viewport.  Usually PGSWIN is called automatically by PGENV,
but it may be called directly by the user.
Arguments:
 X1      (input)  : the x-coordinate of the bottom left corner
                    of the viewport.
 X2      (input)  : the x-coordinate of the top right corner
                    of the viewport (note X2 may be less than X1).
 Y1      (input)  : the y-coordinate of the bottom left corner
                    of the viewport.
 Y2      (input)  : the y-coordinate of the top right corner
                    of the viewport (note Y2 may be less than Y1).
```

A.81 PGTBOX – Draw a box and optionally write HH MM SS style numeric labelling.

```
      SUBROUTINE PGTBOX (XOPT, XTICKD, NXSUBD, YOPT, YTICKD, NYSUBD)
      REAL XTICKD, YTICKD
      INTEGER NXSUBD, NYSUBD
      CHARACTER XOPT*(*), YOPT*(*)
Draw a box and optionally label one or both axes with HH MM SS style
numeric labels (useful for time or RA - DEC plots). Should deal with
axes that increase or decrease, and are positive or negative, or both.
If the HH MM SS labelling is desired, then PGSWIN should have been
previously called with the extrema in SECONDS.
Inputs:
 XOPT   :  X-options for PGBOX.  Same as for PGBOX plus 'Z' for time
           labelling, and 'F' means write only the last part of the
           label for the first time tick on the axis.  E.g., if
           the full first label is 17 42 34.4 then write only 34.4
 YOPT   :  Y-options for PGBOX.  Same as for PGBOX plus 'Z' and 'F'
 XTICKD :  X-axis major tick increment.  Use 0.0 to get default.
 YTICKD :  Y-axis major tick increment.  Use 0.0 to get default.
 NXSUB  :  Number of intervals for minor ticks on X-axis. Use 0 for default
 NYSUB  :  Number of intervals for minor ticks on Y-axis. Use 0 for default
```

A.82 PGTEXT – write text (horizontal, left-justified)

```
      SUBROUTINE PGTEXT (X, Y, TEXT)
      REAL X, Y
      CHARACTER*(*) TEXT
Write text. The bottom left corner of the first character is placed
at the specified position, and the text is written horizontally.
This is a simplified interface to the primitive routine PGPTXT.
For non-horizontal text, use PGPTXT.
Arguments:
 X      (input)  : world x-coordinate of start of string.
 Y      (input)  : world y-coordinate of start of string.
 TEXT   (input)  : the character string to be plotted.
```

A.83 PGUPDT – update display

```
      SUBROUTINE PGUPDT
Update the graphics display: flush any pending commands to the
output device. This routine empties the buffer created by PGBBUF,
but it does not alter the PGBBUF/PGEBUF counter. The routine should
be called when it is essential that the display be completely up to
date (before interaction with the user, for example) but it is not
known if output is being buffered.
Arguments: none
```

A.84 PGVPORT – non-standard alias for PGSVP

```
      SUBROUTINE PGVPORT (XLEFT, XRIGHT, YBOT, YTOP)
      REAL XLEFT, XRIGHT, YBOT, YTOP
```

A.85 PGVSIZ – set viewport (inches)

```
      SUBROUTINE PGVSIZ (XLEFT, XRIGHT, YBOT, YTOP)
      REAL XLEFT, XRIGHT, YBOT, YTOP
Change the size and position of the viewport, specifying
the viewport in physical device coordinates (inches).  The
viewport is the rectangle on the view surface "through"
which one views the graph.  All the PG routines which plot lines
etc. plot them within the viewport, and lines are truncated at
the edge of the viewport (except for axes, labels etc drawn with
PGBOX or PGLAB).  The region of world space (the coordinate
space of the graph) which is visible through the viewport is
specified by a call to PGSWIN.  It is legal to request a
viewport larger than the view surface; only the part which
appears on the view surface will be plotted.
Arguments:
 XLEFT  (input)  : x-coordinate of left hand edge of viewport, in
                   inches from left edge of view surface.
 XRIGHT (input)  : x-coordinate of right hand edge of viewport, in
                   inches from left edge of view surface.
 YBOT   (input)  : y-coordinate of bottom edge of viewport, in
                   inches from bottom of view surface.
 YTOP   (input)  : y-coordinate of top  edge of viewport, in inches
                   from bottom of view surface.
```

A.86 PGVSIZE – non-standard alias for PGVSIZ

```
      SUBROUTINE PGVSIZE (XLEFT, XRIGHT, YBOT, YTOP)
      REAL XLEFT, XRIGHT, YBOT, YTOP
```

A.87 PGVSTAND – non-standard alias for PGVSTD

```
      SUBROUTINE PGVSTAND
```

A.88 PGVSTD – set standard (default) viewport

```
      SUBROUTINE PGVSTD
Define the viewport to be the standard viewport.  The standard
viewport is the full area of the view surface (or subpage),
less a margin of 4 character heights all round for labelling.
It thus depends on the current character size, set by PGSCH.
Arguments: none.
```

A.89 PGWINDOW – non-standard alias for PGSWIN

```
      SUBROUTINE PGWINDOW (X1, X2, Y1, Y2)
      REAL X1, X2, Y1, Y2
```

A.90 PGWNAD – set window and adjust viewport to same aspect ratio

```
      SUBROUTINE PGWNAD (X1, X2, Y1, Y2)
      REAL X1, X2, Y1, Y2
Change the window in world coordinate space that is to be mapped on
to the viewport, and simultaneously adjust the viewport so that the
world-coordinate scales are equal in x and y. The new viewport is
the largest one that can fit within the previously set viewport
while retaining the required aspect ratio.
Arguments:
 X1     (input)  : the x-coordinate of the bottom left corner
                   of the viewport.
 X2     (input)  : the x-coordinate of the top right corner
                   of the viewport (note X2 may be less than X1).
 Y1     (input)  : the y-coordinate of the bottom left corner
                   of the viewport.
 Y2     (input)  : the y-coordinate of the top right corner of the
                   viewport (note Y2 may be less than Y1).
```

Appendix B

PGPLOT SYMBOLS

The PGPLOT primitive subroutines `PGPOINT` and `PGPTEXT` are used for drawing symbols (graph markers) and strings of symbols (text). The available symbols are listed in this appendix. The total number of different symbols available is about 1000. Each symbol is composed of a set of vectors, based on digitized type fonts devised by A. V. Hershey of the US Naval Postgraduate School, and each symbol is assigned a number in the range 0–4000. PGPLOT uses the ASCII code for representing graph markers and symbols. ASCII codes 0–31 are used for the graph markers, and codes 32–127 are used for the standard ASCII printable characters. The font-switching and escape mechanisms described in Chapters 4 and 5 can be used to enlarge the available character set.

Table B.1 shows the graphical representation of all the available symbols arranged according to Hershey's numerical sequence; the blank spaces in this table represent "space" characters of various widths. Note that not every number has an associated character. Any character can be inserted in a text string using an escape sequence of the form `\(`n`)`, where n is the 4-digit Hershey number.

Table B.1 PGPLOT Character Set

0001 A	0002 B	0003 C	0004 D	0005 E	0006 F	0007 G	0008 H	0009 I	0010 J
0011 K	0012 L	0013 M	0014 N	0015 O	0016 P	0017 Q	0018 R	0019 S	0020 T
0021 U	0022 V	0023 W	0024 X	0025 Y	0026 Z	0027 Α	0028 Β	0029 Γ	0030 Δ
0031 Ε	0032 Ζ	0033 Η	0034 Θ	0035 Ι	0036 Κ	0037 Λ	0038 Μ	0039 Ν	0040 Ξ
0041 Ο	0042 Π	0043 Ρ	0044 Σ	0045 Τ	0046 Υ	0047 Φ	0048 Χ	0049 Ψ	0050 Ω
0197	0198	0199	0200 0	0201 1	0202 2	0203 3	0204 4	0205 5	0206 6
0207 7	0208 8	0209 9	0210 .	0211 ,	0212 :	0213 ;	0214 !	0215 ?	0216 '
0217 "	0218 °	0219 $	0220 /	0221 (	0222)	0223 |	0224 −	0225 +	0226 =
0227 ×	0228 *	0229 ·	0230 ‘	0231 ’	0232 →	0233 #	0234 &	0235 ¤	0236 ●
0238 ‥	0239 ‥	0240 —	0242 ⌣	0248 ≈	0250 ≅	0252 ⇌	0254 □	0256 •	0258 —
0259 —	0261 ½	0262 ⅓	0263 ⅙	0264 ⅛	0265 ⅔	0266 ⅜	0267 ⅝	0268 ⅞	0269 ⅚
0270 ¼	0271 ¾	0272 £	0273 ®	0274 ©	0275 ≢	0276 ⋯	0278 ↔	0279 ↕	0280 ¢
0281 Ð	0282 ∍	0284 |||	0501 A	0502 B	0503 C	0504 D	0505 E	0506 F	0507 G
0508 H	0509 I	0510 J	0511 K	0512 L	0513 M	0514 N	0515 O	0516 P	0517 Q
0518 R	0519 S	0520 T	0521 U	0522 V	0523 W	0524 X	0525 Y	0526 Z	0527 A

Table B.1 (continued)

0528 B	0529 Γ	0530 Δ	0531 E	0532 Z	0533 H	0534 Θ	0535 I	0536 K	0537 Λ
0538 M	0539 N	0540 Ξ	0541 O	0542 Π	0543 P	0544 Σ	0545 T	0546 ϒ	0547 Φ
0548 X	0549 Ψ	0550 Ω	0551 $\mathcal{A}$	0552 $\mathcal{B}$	0553 $\mathcal{C}$	0554 $\mathcal{D}$	0555 $\mathcal{E}$	0556 $\mathcal{F}$	0557 $\mathcal{G}$
0558 $\mathcal{H}$	0559 $\mathcal{I}$	0560 $\mathcal{J}$	0561 $\mathcal{K}$	0562 $\mathcal{L}$	0563 $\mathcal{M}$	0564 $\mathcal{N}$	0565 $\mathcal{O}$	0566 $\mathcal{P}$	0567 $\mathcal{Q}$
0568 $\mathcal{R}$	0569 $\mathcal{S}$	0570 $\mathcal{T}$	0571 $\mathcal{U}$	0572 $\mathcal{V}$	0573 $\mathcal{W}$	0574 $\mathcal{X}$	0575 $\mathcal{Y}$	0576 $\mathcal{Z}$	0583 ∇
0590 _	0601 a	0602 b	0603 c	0604 d	0605 e	0606 f	0607 g	0608 h	0609 i
0610 j	0611 k	0612 l	0613 m	0614 n	0615 o	0616 p	0617 q	0618 r	0619 s
0620 t	0621 u	0622 v	0623 w	0624 x	0625 y	0626 z	0627 α	0628 β	0629 γ
0630 δ	0631 ε	0632 ζ	0633 η	0634 ϑ	0635 ι	0636 κ	0637 λ	0638 μ	0639 ν
0640 ξ	0641 o	0642 π	0643 ρ	0644 σ	0645 τ	0646 υ	0647 φ	0648 χ	0649 ψ
0650 ω	0651 $\mathscr{a}$	0652 $\mathscr{b}$	0653 $\mathscr{c}$	0654 $\mathscr{d}$	0655 $\mathscr{e}$	0656 $\mathscr{f}$	0657 $\mathscr{g}$	0658 $\mathscr{h}$	0659 $\mathscr{i}$
0660 $\mathscr{j}$	0661 $\mathscr{k}$	0662 $\mathscr{l}$	0663 $\mathscr{m}$	0664 $\mathscr{n}$	0665 $\mathscr{o}$	0666 $\mathscr{p}$	0667 $\mathscr{q}$	0668 $\mathscr{r}$	0669 $\mathscr{s}$
0670 $\mathscr{t}$	0671 $\mathscr{u}$	0672 $\mathscr{v}$	0673 $\mathscr{w}$	0674 $\mathscr{x}$	0675 $\mathscr{y}$	0676 $\mathscr{z}$	0677 ℓ	0683 ∂	0684 ϵ
0685 θ	0686 ϕ	0687 ς	0697	0698	0699	0700 0	0701 1	0702 2	0703 3
0704 4	0705 5	0706 6	0707 7	0708 8	0709 9	0710 .	0711 ,	0712 :	0713 ;

Table B.1 (continued)

0714 !	0715 ?	0716 '	0717 "	0718 °	0719 $	0720 /	0721 (	0722)	0723 |
0724 −	0725 +	0726 =	0727 ×	0728 *	0729 ·	0730 ‘	0731 ’	0732 →	0733 #
0734 &	0735	0737 ‖	0738 ⊥	0739 ∠	0740 ∴	0741 ♤	0742 ♡	0743 ◇	0744 ♧
0745	0746 ⚜	0750 ,	0751 •	0752 *	0753 ▲	0754	0755	0756 ^	0757
0758 ⌒	0759 ◡	0760 ‿	0761 ›	0762	0763	0764	0765 ∼	0766 ∞	0767
0768	0796 —	0797 /	0798 |	0799 \	0800 —	0801 /	0802 /	0803 |	0804 \
0805 \	0806 −	0807 /	0808 |	0809 \	0810	0811	0812	0813	0814 ‿
0815 (	0816)	0817 ⌒	0818	0819	0820	0821	0822	0823	0824
0825	0826	0827	0828 •	0829 ›	0830 ---	0831	0832 ∧	0833	0834 ▽
0840 ○	0841 □	0842 △	0843 ◊	0844 ☆	0845 +	0846 ×	0847 *	0850 ●	0851 ■
0852 ▲	0853 ◀	0854 ▼	0855 ▶	0856 ★	0857	0860 ⚓	0861	0862 ⚒	0863
0864	0865	0866 ✚	0867 ©	0868 ✡	0869	0870	0871	0872	0873
0874	0899 ·	0900 ∘	0901 ○	0902 ○	0903 ○	0904 ○	0905 ○	0906 ○	0907 ○
0908	0909	0910 ¢	2001 A	2002 B	2003 C	2004 D	2005 E	2006 F	2007 G
2008 H	2009 I	2010 J	2011 K	2012 L	2013 M	2014 N	2015 O	2016 P	2017 Q

Table B.1 (continued)

2018 R	2019 S	2020 T	2021 U	2022 V	2023 W	2024 X	2025 Y	2026 Z	2027 A
2028 B	2029 Γ	2030 Δ	2031 E	2032 Z	2033 H	2034 Θ	2035 I	2036 K	2037 Λ
2038 M	2039 N	2040 Ξ	2041 O	2042 Π	2043 P	2044 Σ	2045 T	2046 Υ	2047 Φ
2048 X	2049 Ψ	2050 Ω	2051 A	2052 B	2053 C	2054 D	2055 E	2056 F	2057 G
2058 H	2059 I	2060 J	2061 K	2062 L	2063 M	2064 N	2065 O	2066 P	2067 Q
2068 R	2069 S	2070 T	2071 U	2072 V	2073 W	2074 X	2075 Y	2076 Z	2077 $\aleph$
2078 Å	2101 a	2102 b	2103 c	2104 d	2105 e	2106 f	2107 g	2108 h	2109 i
2110 j	2111 k	2112 l	2113 m	2114 n	2115 o	2116 p	2117 q	2118 r	2119 s
2120 t	2121 u	2122 v	2123 w	2124 x	2125 y	2126 z	2127 α	2128 β	2129 γ
2130 δ	2131 ε	2132 ζ	2133 η	2134 ϑ	2135 ι	2136 κ	2137 λ	2138 μ	2139 ν
2140 ξ	2141 o	2142 π	2143 ρ	2144 σ	2145 τ	2146 υ	2147 φ	2148 χ	2149 ψ
2150 ω	2151 a	2152 b	2153 c	2154 d	2155 e	2156 f	2157 g	2158 h	2159 i
2160 j	2161 k	2162 l	2163 m	2164 n	2165 o	2166 p	2167 q	2168 r	2169 s
2170 t	2171 u	2172 v	2173 w	2174 x	2175 y	2176 z	2177 ff	2178 fi	2179 fl
2180 ffi	2181 ffl	2182 ı	2184 ϵ	2185 θ	2186 ϕ	2187 ς	2190 $\wp$	2191 *ff*	2192 *fi*

Table B.1 (continued)

2193 ﬂ	2194 ﬃ	2195 ﬄ	2196 ı	2197	2198	2199	2200 0	2201 1	2202 2
2203 3	2204 4	2205 5	2206 6	2207 7	2208 8	2209 9	2210 .	2211 ,	2212 :
2213 ;	2214 !	2215 ?	2216 ′	2217 ″	2218 °	2219 *	2220 /	2221 (	2222)
2223 [	2224]	2225 {	2226 }	2227 ⟨	2228 ⟩	2229 ∣	2230 ‖	2231 −	2232 +
2233 ±	2234 ∓	2235 ×	2236 ·	2237 ÷	2238 =	2239 ≠	2240 ≡	2241 <	2242 >
2243 ≦	2244 ≧	2245 ∝	2246 ∼	2247 ˆ	2248 ´	2249 `	2250 ˘	2251 ’	2252 ‘
2253 ‘	2254 ’	2255 √	2256 ⊂	2257 ∪	2258 ⊃	2259 ∩	2260 ∈	2261 →	2262 ↑
2263 ←	2264 ↓	2265 ∂	2266 ∇	2267 √	2268 ∫	2269 ∮	2270 ∞	2271 %	2272 &
2273 @	2274 $	2275 #	2276 §	2277 †	2278 ‡	2279 ∃	2281 ☉	2282 ☿	2283 ♀
2284 ⊕	2285 ♂	2286 ♃	2287 ♄	2288 ♅	2289 ♆	2290 ♇	2291 ☾	2292 ☄	2293 ✳
2294 ☊	2295 ☋	2296 ↗	2297 ↙	2298 ↖	2299 ↘	2301 ♈	2302 ♉	2303 ♊	2304 ♋
2305 ♌	2306 ♍	2307 ♎	2308 ♏	2309 ♐	2310 ♑	2311 ♒	2312 ♓	2317 ·	2318 𝅮
2319 𝅮	2320 𝅝	2321 𝅗	2322 ●	2323 ♯	2324 ♮	2325 ♭	2326 𝄻	2327 𝄼	2328 𝄪
2329 𝄾	2330 𝄞	2331 𝄢	2332 𝄡	2367 ·	2368 𝅮	2369 𝅮	2370 𝅝	2371 𝅗	2372 ●
2373 ♯	2374 ♮	2375 ♭	2376 𝄻	2377 𝄼	2378 𝄽	2379 𝄾	2380 𝄞	2381 𝄢	2382 𝄡

Table B.1 (continued)

2401 Π	2402 Σ	2403 (	2404)	2405 [	2406]	2407 {	2408 }	2409)	2410 (
2411 √	2412 ∫	2501 A	2502 B	2503 C	2504 D	2505 E	2506 F	2507 G	2508 H
2509 I	2510 J	2511 K	2512 L	2513 M	2514 N	2515 O	2516 P	2517 Q	2518 R
2519 S	2520 T	2521 U	2522 V	2523 W	2524 X	2525 Y	2526 Z	2551 $\mathcal{A}$	2552 $\mathcal{B}$
2553 $\mathcal{C}$	2554 $\mathcal{D}$	2555 $\mathcal{E}$	2556 $\mathcal{F}$	2557 $\mathcal{G}$	2558 $\mathcal{H}$	2559 $\mathcal{I}$	2560 $\mathcal{J}$	2561 $\mathcal{K}$	2562 $\mathcal{L}$
2563 $\mathcal{M}$	2564 $\mathcal{N}$	2565 $\mathcal{O}$	2566 $\mathcal{P}$	2567 $\mathcal{Q}$	2568 $\mathcal{R}$	2569 $\mathcal{S}$	2570 $\mathcal{T}$	2571 $\mathcal{U}$	2572 $\mathcal{V}$
2573 $\mathcal{W}$	2574 $\mathcal{X}$	2575 $\mathcal{Y}$	2576 $\mathcal{Z}$	2601 a	2602 b	2603 c	2604 d	2605 e	2606 f
2607 g	2608 h	2609 i	2610 j	2611 k	2612 l	2613 m	2614 n	2615 o	2616 p
2617 q	2618 r	2619 s	2620 t	2621 u	2622 v	2623 w	2624 x	2625 y	2626 z
2651 *a*	2652 *b*	2653 *c*	2654 *d*	2655 *e*	2656 *f*	2657 *g*	2658 *h*	2659 *i*	2660 *j*
2661 *k*	2662 *l*	2663 *m*	2664 *n*	2665 *o*	2666 *p*	2667 *q*	2668 *r*	2669 *s*	2670 *t*
2671 *u*	2672 *v*	2673 *w*	2674 *x*	2675 *y*	2676 *z*	2697	2698	2699	2700 0
2701 1	2702 2	2703 3	2704 4	2705 5	2706 6	2707 7	2708 8	2709 9	2710 .
2711 ,	2712 :	2713 ;	2714 !	2715 ?	2716 ‘	2717 ’	2718 &	2719 $\$$	2720 /
2721 (	2722)	2723 *	2724 −	2725 +	2726 =	2727 ′	2728 ″	2729 °	2747

Table B.1 (continued)

2748	2749	2750 0	2751 1	2752 2	2753 3	2754 4	2755 5	2756 6	2757 7
2758 8	2759 9	2760 .	2761 ,	2762 :	2763 ;	2764 !	2765 ?	2766 ‘	2767 ’
2768 &	2769 $	2770 /	2771 (	2772)	2773 *	2774 −	2775 +	2776 =	2777 ′
2778 ″	2779 °	2801 А	2802 Б	2803 В	2804 Г	2805 Д	2806 Е	2807 Ж	2808 З
2809 И	2810 Й	2811 К	2812 Л	2813 М	2814 Н	2815 О	2816 П	2817 Р	2818 С
2819 Т	2820 У	2821 Ф	2822 Х	2823 Ц	2824 Ч	2825 Ш	2826 Щ	2827 Ъ	2828 Ы
2829 Ь	2830 Э	2831 Ю	2832 Я	2901 а	2902 б	2903 в	2904 г	2905 д	2906 е
2907 ж	2908 з	2909 и	2910 й	2911 к	2912 л	2913 м	2914 н	2915 о	2916 п
2917 р	2918 с	2919 т	2920 у	2921 ф	2922 х	2923 ц	2924 ч	2925 ш	2926 щ
2927 ъ	2928 ы	2929 ь	2930 э	2931 ю	2932 я				

Appendix C1

INSTALLATION INSTRUCTIONS (VMS)

C1.1 Introduction

PGPLOT consists of four “layers” of subroutines. The subroutines in each layer call only subroutines in the same layer or in the next layer down:

1. The top-level PGPLOT subroutines; these routines call only other PGPLOT routines.
2. The “primitive” PGPLOT routines; these are used by the top-level routines and can be called by users’ programs. They call the routines at the next level to perform graphical output.
3. Device-independent support routines. These are responsible for things like scaling, windowing, and character-generation.
4. Device handler routines: these generate graphical output for specific device types. There is one device handler subroutine for each supported device type.

Routines at levels 1 and 2 are the only ones that should be referenced directly by a user’s program. They are all described fully in Appendix A.

C1.2 Restoring the Save Set

PGPLOT is usually distributed in the form of a VAX/VMS BACKUP save set on magnetic tape. This save set contains a directory tree: the main directory, [PGPLOT], and several subdirectories: [PGPLOT.DRIVERS], [PGPLOT.EXAMPLES], [PGPLOT.FONTS], [PGPLOT.MANUAL], and [PGPLOT.SOURCE]. The save set should be restored to a similar directory tree. The following example indicates how the VMS BACKUP command can be used to copy the files into directories called [USER.PGPLOT] and [USER.PGPLOT.*]. It is a good idea to make sure that these destination directories either do not exist or are empty before beginning the BACKUP command.

```
$ MOUNT/FOREIGN MTA0: PGPLOT
$ BACKUP/LOG/VERIFY MTA0:PGPLOT.BCK/SELECT=[PGPLOT...] -
        DISK:[USER.PGPLOT...]
$ DISMOUNT MTA0:
```

The main directory, [PGPLOT], contains all the files needed to link and run PGPLOT programs. The subdirectories contain the source code, example programs, utility programs, and documentation.

The directory [PGPLOT] includes:

1. Object code: the object-module library GRPCKG.OLB, the shareable image GRPSHR.EXE, and the shareable-image symbol-table library GRPSHR.OLB.
2. Command procedures: LOGICAL.COM and MAKEHELP.COM. These procedures are described below.
3. Documentation: PGPLOT.DOC is the text of the subroutine synopses in Appendix A. PGPLOT.HLB is a VMS-format HELP library containing the same information.
4. Miscellaneous: DOCUMENT.FOR, DOCUMENT.EXE (used in creating documentation); GRFONT.DAT (binary representation of the PGPLOT character set); GMFPLOT.FOR and GMFPLOT.EXE (metafile translator); PGATTRIB.FOR (a Fortran include file defining symbolic names for the various PGPLOT attribute codes).

The directory [PGPLOT.DRIVERS] contains source code for device-handler subroutines that are *not* included in the executable version of PGPLOT as distributed. Instructions for incorporating these device handlers in the executable version are given below. Adding support for a new device type requires the addition of a new device handler subroutine. No other changes to PGPLOT are required. For details, consult Appendix E.

The directory [PGPLOT.EXAMPLES] contains a number of programs for demonstrating and testing PGPLOT. The source code is in files PGEX1-.FOR, PGEX2.FOR, etc., and executable code in PGEX1.EXE, etc.. The command procedure EXCOMP.COM can be used to recompile and link one or all of the example programs.

The directory [PGPLOT.FONTS] contains some utility programs for manipulating the binary file containing character definitions (font information). This directory can be deleted if you do not want to edit the character set.

The directory [PGPLOT.MANUAL] contains the text of this PGPLOT manual (in TEX format) and Fortran programs for creating the figures. This directory can be deleted if you do not wish to print copies of the manual, or if you do not have the TEX document formatting program.

The directory [PGPLOT.SOURCE] contains the complete source code for PGPLOT, including the device handlers built in to the distribution version. PGPLOT.FOR contains the Fortran source code for routines in levels 1 and 2; all these routines have names beginning with PG. GRPCKG.FOR, GRCHAR.FOR, GRSYMB.FOR, and GRVMS.FOR contain the source code for routines in level 3. The file GRVMS.FOR contains *system-dependent* routines which will have to be rewritten for a different operating system. The various files *DRIVER.FOR contain the source code for the level-4 device handlers. All the routines in level 3 have names beginning with GR. The files PGPLOT.INC and GRPCKG1.INC are referenced by VAX Fortran INCLUDE statements in PGPLOT.FOR and GRPCKG.FOR, respectively. The command procedures ADD.COM, BUILD.COM, COMPILE.COM, and NEWGE.COM are used to rebuild PGPLOT from the source code (see below).

C1.3 Logical Names

A number of VMS logical names must be defined before PGPLOT can be used. The procedure LOGICAL.COM will define these logical names, but it may be necessary to modify it slightly. A possible definition is the following:

```
$ DEFINE/SYSTEM PGPLOT_DIR   SYS:[PGPLOT]
$ DEFINE/SYSTEM LNK$LIBRARY  PGPLOT_DIR:GRPSHR.OLB
$ DEFINE/SYSTEM PGPLOT_TYPE  PRINTRONIX
$ DEFINE/SYSTEM PGPLOT_FONT  PGPLOT_DIR:GRFONT.DAT
$ DEFINE/SYSTEM GRPSHR       PGPLOT_DIR:GRPSHR.EXE
```

It is not essential that they be "system" logical names.

1. `PGPLOT_DIR`. This is a logical name pointing to the directory containing the PGPLOT files. The name need not be `PGPLOT_DIR`; another name can be chosen, so long as the other logical names are modified accordingly.
2. `LNK$LIBRARY`. This is optional. It tells the VMS linker to automatically scan GRPSHR.OLB for subroutine references. If `LNK$LIBRARY` is already defined, choose another name like `LNK$LIBRARY_1` (see the VAX/VMS Linker manual).
3. `PGPLOT_TYPE`. This is optional. It provides a default device type for use by PGPLOT when no device type is provided explicitly by the user.
4. `PGPLOT_FONT`. This is essential. Whenever a PGPLOT program is run, it attempts to read the character-set definition from the file defined by this logical name. (If it fails, the program will continue, but requests to plot text or graph markers will be ignored.)
5. `GRPSHR`. This logical name must be defined at run-time if the shareable-image version of PGPLOT is used (see below).

C1.4 The Shareable Image

Programs using the PGPLOT library can be linked in two different ways:

```
$ LINK EXAMPLE,PGPLOT_DIR:GRPSHR/LIB
$ LINK EXAMPLE,PGPLOT_DIR:GRPCKG/LIB
```

Usually the first library (`GRPSHR.OLB`) should be used. When this library is used, the subroutines are not included in the `.EXE` file, but are fetched from a *shareable image* when you execute the `RUN` command. This makes the `.EXE` file much smaller, and means that the program need not be relinked when changes are made to the graphics subroutines; but the `.EXE` file can only be run on a machine which has a copy of the shareable image. If the second library (`GRPCKG.OLB`) is used, the subroutines are included in the `.EXE` file and the program can be run on any VAX computer running a compatible version of VMS. In general, the shareable image is to be preferred.

C1.5 Recompiling PGPLOT

To recompile PGPLOT, proceed as follows.

1. In directory `[PGPLOT.SOURCE]`: make sure that the source code for all the device handlers you wish to include is in this directory (files `*DRIVER.FOR`). Device handlers that you do not wish to include should be moved to directory `[PGPLOT.DRIVERS]`.
2. Execute the DCL command procedure `COMPILE.COM`. This compiles all the Fortran files in the directory, and creates a new version of the object-module library `GRPCKG.OLB`. See the comments in the procedure for more information.
3. The shareable image, `GRPSHR.EXE`, and shareable-image symbol-table library, `GRPSHR.OLB`, can be rebuilt from `GRPCKG.OLB` by executing the command procedure `BUILD.COM`. Again, see the comments in the procedure for more information. (Note: do not modify the symbol table defined in `BUILD.COM`; if you do, the resulting programs will not be portable.)
4. You can now delete all the object module files (`*.OBJ`) if you wish.
5. The new version of PGPLOT consists of three files: `GRPCKG.OLB`, `GRPSHR.EXE`, and `GRPSHR.OLB`. It is possible to test the new version by redefining logical name `GRPSHR` to point to the new `GRPSHR.EXE`. If you are satisfied with the new version, rename (or copy) these three files into the main directory, `[PGPLOT]`, superseding the old version.

C1.6 Recompiling the Example Programs

The procedure `EXCOMP.COM` in directory `[PGPLOT.EXAMPLES]` can be used to compile and link any of the example programs:

```
$ @EXCOMP 7
$ @EXCOMP
```

The first command compiles PGEX7. The second compiles *all* the example programs.

C1.7 Rebuilding the Documentation Files

The files `PGPLOT.DOC` and `PGPLOT.HLB` contain documentation extracted from the PGPLOT source code. The files can be rebuilt by executing the procedure `MAKEHELP.COM` in directory `[PGPLOT]`. This procedure also updates the TEX version of the documentation included in Appendix A of the manual.

C1.8 Printing the Manual

The TEX source-code for the PGPLOT manual can be found in the subdirectory `[PGPLOT.MANUAL]` in a number of files with type `.TEX`. These files can be used to make a printable copy of the manual if you have the TEX text-formatting program:

```
$ TEX PGPLOT
```

The file `PGPLOT.TEX` contains TEX `\input` commands that reference the other `.TEX` files. Please do not modify the TEX files: send your comments and bug reports to the author.

The figures in the manual were generated using PGPLOT subroutines. The Fortran programs used to generate them can also be found in the `[PGPLOT.MANUAL]` directory.

C1.9 Adding a new PGPLOT routine

The author is happy to receive contributions of new high-level routines for inclusion in PGPLOT. Some general guidelines should be observed when preparing such routines:

1. The routine should be written (as far as possible) in standard Fortran-77.
2. The routine should call only other PGPLOT routines, or special support routines provided along with it. It should not use routines from other libraries (especially commercial ones!). Note that if it is to be installed in the shareable library, it cannot use common blocks for communication with the program that calls it. It may, if necessary, use common blocks to communicate with other PGPLOT routines, but this is discouraged.
3. It should be modular: that is, it should avoid, as far as possible, making assumptions about the environment it is to run in, or modifying that environment. In particular: if it changes any attribute (*e.g.*, color index) it should save the previous value of that attribute (obtained by calling a `PGQ...` routine) and reset it before exiting; it should not change the window or viewport, but if it must, it should restore the old values before exiting; it should not call `PGBEGIN` or `PGEND`.
4. It should call routine `PGBBUF` at the beginning, and `PGEBUF` before exiting, in order to buffer its output.
5. It should provide a generally useful function rather than a highly specific one, with as many parameters as possible accessible to the user (*e.g.*, as arguments) rather than fixed in the program.

Appendix C2

INSTALLATION INSTRUCTIONS (UNIX)

C2.1 Introduction

All UNIX systems are different, so the installation procedure described here may not work on your system, and some editing and debugging may be required. Please read all the installation instructions before attempting to install PGPLOT. I first describe the installation procedure which I have used on our systems (Convex and Sun); I hope it will work on other Berkeley-derived UNIX systems. If you have trouble with the installation, you should read Section C.3, which is intended to provide sufficient background information for you to adapt PGPLOT to a different system.

The UNIX version of PGPLOT is distributed in source form on a 'tar' tape (usually a 9-track, 1600-bpi tape).

C2.2 Basic Installation

The following installation procedure has been run on a Convex C-1 running Version 6.2 or 7.0 of the Convex operating system with Fortran compiler FC Version 4.1 or 5.0, and on Sun-3 and Sun-4 workstations running SunOS Release 4.0 with Sun Fortran Release 1.1. It does *not* work with earlier versions of the Convex Fortran compiler, the Sun operating system, or the Sun Fortran compiler.

1. Copy the files from the distribution tape to your disk. The files are organized in three directories: `pgplot`, `pgplot/examples`, and `pgplot/fonts`. To copy the files from the 'tar' tape, you should first change your default directory to the *parent* directory of `pgplot`, and use a command like the following:

```
tar xv pgplot
```

This will create a `pgplot` directory in your current directory and install the PGPLOT files in it. (If you already have a `pgplot` directory, remove it first.) If you are not using 'tar' but are copying the files from another system, make sure that you put the files in the correct directories.

2. Compile the PGPLOT subroutines and create the PGPLOT object library. Change your default directory to PGPLOT and then use 'make' to do the compilation. The distribution tape contains two 'makefiles', one for Convex (`Makefile.CONVEX`) and one for Sun (`Makefile.SUN`). You can rename the appropriate file to `Makefile`, or use the `-f` option of 'make':

```
cd pgplot
make -f Makefile.SUN
```

If all goes well, you should get no error messages from the compilation. (The SUN Fortran compiler issues an incorrect warning message that `MOD` is an unused variable when compiling `grre04` and `grte04`; you can ignore this message.)

3. Build the binary font file. The font file (read at run time by PGPLOT programs) is distributed in an ASCII text form in file `grfont.txt` and must be converted to binary form before use. This is done using the 'makefile' in the `fonts` directory; again there are separate versions for Convex and Sun:

```
cd fonts
make -f Makefile.SUN grfont.dat
```

This compiles a program `pgpack` to do the conversion and then runs it. It should display a message like the following:

```
Characters defined:   996
Array cells used:   26732
```

You now need to define an environment variable that defines the location of the binary font file, e.g.:

```
setenv PGPLOT_FONT /usr/name/pgplot/fonts/grfont.dat
```

Substitute the appropriate path for `/usr/name/pgplot`; or while you are in the `fonts` directory, type

```
setenv  PGPLOT_FONT `pwd`/grfont.dat
```

The `fonts` directory also contains utility programs `pgunpack` and `pgdchar` that you will need only if you plan to modify the font file (not recommended).

4. Compile the test and demonstration programs in the `examples` directory. Again there there are separate 'makefiles' for Convex and Sun:

```
cd ../examples
make -f Makefile.SUN
```

At this point, you may run out of disk space (if you haven't done already) as the executable programs are rather large. If necessary, you can compile the programs one at a time, e.g.:

```
make -f Makefile.SUN pgdemo1
```

to compile program `pgdemo1`. The two demonstration programs `pgdemo1` and `pgdemo2` each generate several screens (pages) illustrating the use of PGPLOT subroutines. There is also a general test program, `pgex17`, described in Appendix E, and two test programs for the cursor routines, `pgex15` and `pgex18`.

5. Run at least one of the test programs, e.g. `pgdemo1`.

```
pgdemo1
```

The program will prompt for a device specification. Type a question mark `?` to get a list of available device handlers. Read the PGPLOT manual for details of device specifications, and the remainder of this Appendix for notes on the available device handlers. Run the program once for each device that you wish to test. I hope that all will go well; if not, read the next section. The commonest problem is that you get graphs with no text. This implies that the program cannot read the font file (you should also get a message to this effect). Check the definition of `PGPLOT_FONT` (see above).

6. If everything so far looks all right, you can move the files to their final destinations (this may require superuser privilege and the cooperation of your system manager). You need to keep two files: the object library `pgplot/libpgplot.a` and the binary font file `pgplot/fonts/grfont.dat`. Everything else can be deleted if you want to conserve disk space; it can always be recovered from the distribution tape. The object library should (ideally) be moved to the directory where the loader expects to find libraries; usually `/usr/lib`:

```
cd pgplot
cp libpgplot.a /usr/lib
```

You can then use `-lpgplot` with `ld` or your Fortran compiler to link a program with the PGPLOT library. The location of the font file is arbitrary, but as distributed, PGPLOT expects to find it in `/usr/local/lib/grfont.dat`:

```
cd fonts
cp grfont.dat /usr/local/lib
```

If you want to put it in some other location, you can *either* define `PGPLOT_FONT` every time you use a PGPLOT program (see above), *or* modify the default location by editing file `pgplot/grsy00.f` and recompiling (see step 2). (Note that if you modify any PGPLOT routine, you will have to remodify it each time you install a new version of PGPLOT.)

7. Finally, you should attempt to compile and link a PGPLOT program of your own. On Convex:

```
fc -o program program.f -lpgplot
```

On Sun:

```
f77 -o program program.f -lpgplot -lsuntool -lcgi77 -lcgi\
    -lsunwindow -lpixrect -lm
```

Note that the Sun version requires all the above libraries to be included, in the order indicated (assuming that the /SUNVIEW and /CGI device handlers are included, as they are in the distribution tape).

C2.3 Advanced Installation

This section addresses problems that you may encounter trying to install PGPLOT on a different UNIX system, and indicates what options are open to you to solve them.

Examine the 'makefiles' in the `pgplot` directory and make sure that you understand what they do. In particular, you may need to change the definitions of the Fortran compiler and compilation switches.

The subroutines are organized into several groups in the 'makefile':

1. PG routines. These are the top level PGPLOT routines that can be called by application programs, plus a few internal routines. They are identical to the VMS version, and are written in standard Fortran-77 (I believe) with two exceptions; the `INCLUDE` statement and the use of subroutines names longer than 6 characters. The former can be avoided by use of a preprocessor or by hand-editing the code to replace each `INCLUDE` statement with the text of the included file; the latter is fundamental to PGPLOT, but you might try truncating names to 6 characters.
2. GR routines. These are internal routines. They are also supposed to be standard Fortran-77, but there are a few problems. As in the PG routines, some subroutine names are longer than 6 characters, and the `INCLUDE` statement is used. A large array `BUFFER` used in `grsy00.f` and `grsyxd.f` is declared `INTEGER*2` to save space both internally and in the binary font file (the array is just a copy of the disk file); if you change it to `INTEGER`, you must make the same change in the `pgpack`

program in the `fonts` directory. Routine `grgrps.f` uses a hexadecimal format code (`Z2.2`) which is non-standard.

3. Drivers. Each device handler is a subroutine with a name like `xxdriv` (for device `xx`). The 'makefile' lists the additional subroutines required for each handler. The `DRIVERS` macro lists all the device handlers to be included in the final library. You customize PGPLOT by changing the list assigned to `DRIVERS`. The device handlers are not written in standard Fortran-77, although I have tried to adhere to the standard where possible. Please let me know what problems you have. Some of the handlers call on subroutines written in C rather than Fortran. The conventions for writing a C subroutine to be called from a Fortran program are different on different systems (it may not even be possible). Thus you may need to do a lot of work to get each handler going. But note that you don't need to have *all* the handlers running in order to test PGPLOT. It is sufficient to include just the null device handler, which is written in standard Fortran-77. (See below for notes on each handler.)
4. Dispatch routine. The routine `grexec` must be modified to include the list of device handlers you wish to include in PGPLOT. Two different versions are provided on the distribution tape; one for Convex and one for Sun. It is a simple multi-way `GOTO` (`CASE` statement), one branch for each handler. Eventually this routine will be assembled automatically during the installation procedure, but at present you must edit it by hand.
5. System routines. These routines, called from all levels of PGPLOT, encapsulate various operating-system dependent functions. If you have trouble with these routines, you will need to write new versions. I hope that the internal comments are sufficient to explain what each routine is supposed to do. The version distributed calls on a number of Fortran library functions which are normally part of Berkeley-UNIX but which may not be present in other varieties (e.g., `getenv`, `getlog`, `hostnm`).
6. Obsolete routines. This group includes a number of routines that are no longer an official part of PGPLOT, but they may be called by some old PGPLOT application programs. If you have such programs, I recommend you rewrite them, but if necessary you can compile an additional library containing these routines by the command `make -f Makefile.SUN libpgobs.a` .

C2.4 Device Handlers

The UNIX version of PGPLOT is distributed with a much smaller set of device handlers than the VMS version. If you modify any of the device handlers or write new ones, please (a) try to keep them as portable as possible, and (b) send copies to me so that I can include them in future distributions. See Appendix E for instructions on writing a device handler.

Null device The null device handler is `nudriv`. Use device specification `/NULL`. Output sent to the null device is discarded; i.e., PGPLOT produces no output. This device handler is fully portable.

SunView (Sun only) File `svdriv` is a first attempt at a handler for Sun workstations running SunView, from Brian M. Sutin (sutin@astro.umd.edu). It is written in C. Programs which use it must be linked with several libraries (in order):

```
-lsuntool -lsunwindow -lpixrect
```

Use device specification `/SUNVIEW`. Please let me know if you have any problems with this handler, or make improvements to it.

Sun-CGI (Sun only) File `cgdriv` is another handler for Sun workstations running SunView, from Allyn Tennant. It only works with color workstations, but could probably be easily modified for monochrome workstations. It calls on the Sun CGI routines, and hence programs which use it must be linked with several libraries (in order):

```
-lcgi77 -lcgi -lsunwindow -lpixrect -lm
```

Use device specification `/CGI`. PGPLOT creates a window on the screen, or uses the whole screen if you are not running SunView. There is no cursor available with this handler. The window is closed when your program exits, so it is not possible to overlay one plot on top of another by using the undocumented `/APPEND` qualifier.

IVAS (Convex only) The handler for the IIS IVAS image display (`ivdriv`, device specification `/IVAS`) calls C routines to do most of the work. It is self-contained (no other libraries are required), but will require a compatible version of the Convex device driver for the interface board.

Imagraph (Convex only) This is a handler for the Imagraph image display (`tvdriv`, device specification `/IMAGRAPH`). The board is an AGC SERIES VME-1280-10 board (IMAgraph Corporation, 800 West Cummings Park, Woburn, Massachusetts 01801), and utilizes a Hitachi HD63484 Advanced CRT Controller (ACRTC) as graphics engine. It is configured with 1024×1024 8-bit pixels (image frame memory) plus a 2-bit pixel overlay. The device handler is self-contained (no other libraries are required), but it will require the Caltech Convex device driver for the interface board (contact Judy Cohen for more information).

Tektronix terminals and emulators The basic Tektronix-4010 handler is `tedriv` (`/TEK`). Extensions for use with Retrographics terminals and GraphOn terminals are provided in files `redriv` (`/RETRO`) and `gfdriv` (`/GF`), respectively. The code for these handlers is fairly standard, with the exception of C routines in `gr_term_io.c` for writing to the terminal. As control characters must be output, the terminal is put in 'cbreak' mode during output. The `tedriv` handler works with the Sun Tektronix emulator `tektool`. The cursor works with these device handlers on the Convex but not on Suns: I would appreciate suggestions for fixing this problem.

PostScript printers The PostScript handlers (`psdriv` for landscape mode, `vpdriv` for portrait mode) adhere fairly closely to standard Fortran-77, and so should be easily ported to different systems. The files that they create do not display correctly in the Sun `psview` PostScript preview program owing to what I assume is a bug in Sun's implementation of PostScript. The PostScript handlers can optionally insert a control-D (end-of-file) character at the beginning and end of the PostScript file, which may be needed by some PostScript printers. To enable this option, define the environment variable `PGPLOT_PS_EOF` (with any value).

REGIS terminals (e.g., DEC VT125) The REGIS device handler is in file `vtdriv`; specify a device type of `/VT125`. It is fairly close to standard Fortran-77. The default output device is the user's terminal, but a disk file can also be specified. The display should be satisfactory on VT125 terminals; later DEC terminals (like VT240, VT340) do not have separate memories for text and graphics, and this leads to problems with programs that interleave text and graphical output. At present, the cursor is not implemented.

Other devices (Convex only) The basic installation procedure installs some additional drivers in the Convex version of PGPLOT, but these have not been tested in other versions. Imagen printers: `imdriv`; Printronix printers: `pxdriv`; QMS printers: `qmdriv` and `vqdriv`; Versatec printers: `vedriv` and `vvdriv`.

C2.5 Special Notes: Sun

The Sun Fortran compiler treats the 'backslash' character (\) as a special escape character in literal character strings. This means that any programs that try to make use of the PGPLOT escape-character mechanism, which also uses backslash, will have problems. Until Sun provides a way to disable this annoying behavior of the compiler, the solution is to replace each occurrence of \ by \\. This does not cause problems within the PGPLOT library (there are no literal \ characters), but it does affect the example programs and many application programs. The 'makefile' in the `examples` directory passes the example programs through the Sun filter `f77cvt` which performs this substitution (among other things).

C2.6 Acknowledgments

I am grateful to Jon Danskin of Convex Computer Corporation for the initial port of PGPLOT to Convex-UNIX, Allyn Tennant (NASA Marshall) for providing the Sun-CGI device handler, Brian Sutin (University of Maryland) for the SunView device handler, Neil Killeen (University of Illinois) for comments on the Sun implementation, and Judy Cohen (Caltech) for providing time on Sun-3 and Sun-4 workstations.

T. J. Pearson
29 May 1989

Appendix D

SUPPORTED DEVICES

D.1 Introduction

This Appendix presents device-specific information for some of the supported devices. Table D.1 shows the devices for which device handlers are available, together with the names by which they are known to PGPLOT. The names of the device types can be abbreviated so long as there is no ambiguity; in most cases, this means the first two letters are sufficient. Each installation of PGPLOT is configured with the devices appropriate for that installation, so not every device is available in every installation of PGPLOT. Some devices are available under VMS only, and others are available under Unix only.

The description of each device is organized under the following headings:

Supported device: description of device.

Device type code: the code to be used (following the /) in a PGPLOT device specification; usually this can be abbreviated to two letters. Some devices can be used in two modes: *landscape* (long axis horizontal) and *portrait* (long axis vertical). Different device type codes are used for the two modes.

Default file or device name: the file or device name that is used by default if none is included in the device specification. (For example, if the device specification is given as `/VERS`, it is expanded to `PGPLOT.VEPLOT/VERS`; if it is given as `UV/VERS`, it is expanded to `UV.VEPLOT/VERS`.)

Default view surface dimensions: Most hardcopy devices print on 11×8.5-inch paper, and the standard usable "view surface" is 10.5×8.0 inches. The dimensions are meaningless for most CRT devices where the image size depends on the size of the monitor.

Resolution: The nominal resolution of the device in pixels/inch. The exact resolution can be ill-defined; *e.g.*, on a pen plotter, it depends on the size of the pen nib.

Color capability: What color indices are available? Can the representation of each color index be changed? Do changes to the representation affect primitive elements that have already been drawn?

Table D.1 Available Devices

Terminals	
GraphOn GO-230 terminal	/GF
Tektronix 4006/4010 storage-tube terminal	/TEK4010
Retrographics modified VT100 terminal (VT640)	/RETRO
DEC VT125, VT240, or VT340 terminal (REGIS)	/VT125
Tektronix 4100-series color terminal	/TK4100
ZSTEM 240/4014 terminal emulators (IBM PC)	/ZSTEM
High-resolution Display Devices	
DeAnza (Gould 8500 low resolution)	/DEanza
Grinnell GMR-270 Image Display System	/GRINNELL
Digisolve Ikon Pixel Engine	/IKon
Liacom Graphic Video Display (GVD-02)	/LIacom
Peritek Corp. VCH-Q frame-buffer video	/PERITEK
Peritek Corp. VCK-Q frame-buffer video	/PK
Sigma Args, 7000 series	/ARgs
Sigma, T5670 terminal	/GOC
Tektronix 4014 (12 bit addressing)	/TFILE
VAX/VMS workstations	/WS
Sun workstations (SunView mode)	/SUNVIEW
Sun workstations (using CGI routines)	/CGI
IIS IVAS image display	/IVAS
Imagraph image display	/IMA
Pen Plotters	
Gould (now Bryans) Colourwriter 6320	/CW6320
HPGL Driver (tested on HP7475A plotter)	/HPGL
HPGL Driver (portrait mode)	/VHPG
Hewlett Packard 7221 pen plotter	/HP7221
Houston Instruments HIDMP pen plotter	/HIDMP
Bruning (fmr Nicolet) Zeta 8 Digital Plotter	/ZEta
Laser Printers	
Hewlett Packard LaserJet, LaserJet+,II	/LJ
PostScript device (landscape mode)	/PS
PostScript device (portrait mode)	/VPS
QUIC devices (QMS 800 etc) (landscape mode)	/QMS
QUIC devices (QMS 800 etc) (portrait mode)	/VQMS
Canon LBP-8/A2 Laser printer (landscape mode)	/BCanon
Canon LBP-8/A2 Laser printer (landscape mode)	/CAnon
Canon LBP-8/A2 Laser printer (portrait mode)	/VCanon
Impress (Imagen) device (landscape mode)	/IMPRESS
Impress (Imagen) device (portrait mode)	/VIMPRESS

Table D.1 (continued)

Dot-matrix Printers

EPSON FX100 printer	/EPSON
EXCL (C.Itoh Megaserve) printer	/EXCL
LA50 and other DEC sixel printers	/LA50
Printronix P300 or P600 printer	/PRINTRONIX
Toshiba "3-in-on"" printer, model P351	/TOSHIBA
Versatec V80 printer (landscape mode)	/VERSATEC
Versatec V80 printer (portrait mode)	/VVERSATEC

Other

Null device (no graphical output)	/NULL
Metafile	/FILE

Input capability: Instructions for the use of the graphics cursor, if one is available.

File format: Description of the file in which the graphic commands are stored. For some devices, the commands are sent directly to the device and no intermediate file is required.

Obtaining hardcopy: When an intermediate file is used, this section gives instructions for sending the graphic commands from the file to the device. *It is important to check the correct procedure with your System Manager before attempting to generate a hardcopy plot. Attempting to print or plot a file on the wrong sort of device will at best waste a large quantity of paper, and may damage the device or crash the VAX.*

D.2 Versatec

Supported device: Versatec V-80 printer/plotter (and compatible models).

Device type code: `/VErsatec` (landscape mode), `/VVersatec` (portrait mode).

Default file name: `PGPLOT.VEPLOT` (landscape mode), `PGPLOT.VVPLOT` (portrait mode).

Default view surface dimensions: 10.5 inches horizontal $\times$ 8.0 inches vertical (landscape mode), 8.0 inches horizontal $\times$ 10.5 inches vertical (portrait mode). These are nominal values; the actual scale may vary, particularly in the dimension parallel to the paper motion.

Resolution: 200 pixels/inch (both dimensions).

Color capability: Color indices 0 (erase) and 1 (black) are supported. Requests for other color indices are converted to 1. It is not possible to change color representation.

Input capability: None.

File format: The file contains variable length records (up to 265 bytes), one record corresponding to one horizontal dot row of the plot. The file has record attributes "No Carriage Control." The first byte in each record is control-D (hexadecimal 04: plot mode specifier). The remaining 8-bit bytes each represent 8 dots, with the most significant bit representing the left-most dot; 1 implies the corresponding dot is to be filled in. Thus the maximum number of dots per line is $264 \times 8 = 2112$, corresponding to 10.56 inches at 200 dots per inch. The vertical spacing of dot rows is also 200 per inch. Plot pages are separated by a record containing a form-feed character only (1 byte, hexadecimal 0C). These files are intended for use with a special driver and printer symbiont which recognize the control-D and send the remainder of the record to the Versatec in plot mode instead of text mode. This is *not* the format expected by the standard VMS driver provided by Versatec.

Obtaining hardcopy: Use the VMS `COPY` command to send the file to a suitable device or use the VMS `PRINT` command if a suitable printer queue has been established. Examples:

```
$ COPY PGPLOT.VEPLOT LVAO:
$ PRINT/NOTIFY/QUEUE=VERSATEC PGPLOT.VEPLOT
```

On DEIMOS:

```
$ PRINT PGPLOT.VEPLOT
```

On PHOBOS:

```
$ VPRINT PGPLOT.VEPLOT
```

On Starlink:

```
$ PRINT/PASSALL/QUEUE=SYS_VERSATEC PGPLOT.VEPLOT
```

D.3 PostScript printers

Supported device: any printer that accepts the PostScript page description language, *e.g.*, the LaserWriter (Apple Computer, Inc.).

Device type code: `/PS` (landscape mode), `/VPS` (portrait mode).

Default file name: `PGPLOT.PSPLOT` (landscape mode), `PGPLOT.VPPLOT` (portrait mode).

Default view surface dimensions: 10.5 inches horizontal $\times$ 7.8 inches vertical (landscape mode), 7.8 inches horizontal $\times$ 10.5 inches vertical (portrait mode).

Resolution: Commands sent to the device use coordinate increments of 0.001 inch, giving an "apparent" resolution of 1000 pixels/inch. The true resolution is device-dependent; *e.g.*, on an Apple LaserWriter it is 300 pixels/inch (in both dimensions).

Color capability: Color indices 0 (erase), 1–13 (black), 14 (light grey), and 15 (dark grey) are supported. Requests for other color indices are converted to 1. It is not possible to change color representation.

Input capability: None.

File format: The file contains variable length records containing PostScript commands. The commands use only printable ASCII characters, and the file can be examined or modified with a text editor.

Obtaining hardcopy: Use the VMS `COPY` command to send the file to a suitable device or use the VMS `PRINT` command if a suitable printer queue has been established. On DEIMOS:

```
$ PRINT PGPLOT.PSPLOT/QUEUE=LW
```

References: (1) Adobe Systems, Inc. *PostScript Language Reference Manual.* Addison-Wesley, Reading, Massachusetts, 1985. (2) Adobe Systems, Inc. *PostScript Language Tutorial and Cookbook.* Addison-Wesley, Reading, Massachusetts, 1985.

D.4 QMS Lasergrafix

Supported device: QMS Lasergrafix 800/1200 laser printers and other printers accepting "QUIC" commands.

Device type code: `/QMs` (landscape mode), `/VQms` (portrait mode).

Default file name: `PGPLOT.QMPLOT` (landscape mode), `PGPLOT.QEPLOT` (portrait mode).

Default view surface dimensions: 10.5 inches horizontal $\times$ 8.0 inches vertical (landscape mode), 8.0 inches horizontal $\times$ 10.5 inches vertical (portrait mode). These are nominal values; the actual scale may vary.

Resolution: 300 pixels/inch (both dimensions). Commands sent to the device use coordinate increments of 0.001 inch, giving an "apparent" resolution of 1000 pixels/inch.

Color capability: Color indices 0 (erase) and 1 (black) are supported. Requests for other color indices are converted to 1. It is not possible to change color representation.

Input capability: None.

File format: The file contains variable length records containing QUIC commands. The commands use only printable ASCII characters, and the file can be examined or modified with a text editor.

Obtaining hardcopy: Use the VMS `COPY` command to send the file to a suitable device or use the VMS `PRINT` command if a suitable printer queue has been established. Examples:

```
$ COPY PGPLOT.QMPLOT TXAO:
$ PRINT/NOTIFY/QUEUE=LASER PGPLOT.QMPLOT
```

On XHMEIA:

```
$ LASER PGPLOT.QMPLOT
```

On CVAX:

```
$ PRINT/QUEUE=TXAO PGPLOT.QMPLOT
```

D.5 Printronix

Supported device: Printronix P300, P600 and equivalent dot-matrix printer-plotters.

Device type code: `/PRintronix` (landscape mode only).

Default file name: `PGPLOT.PRPLOT`.

Default view surface dimensions: 13.2 inches horizontal $\times$ 10.0 inches vertical, on standard 15 $\times$ 11.0-inch computer paper. These are nominal values, but the printer is usually quite accurately aligned.

Resolution: 60 pixels/inch horizontal $\times$ 72 pixels/inch vertical.

Color capability: Color indices 0 (erase) and 1 (black) are supported. Requests for other color indices are converted to 1. It is not possible to change color representation.

Input capability: None.

File format: A Printronix file contains variable length records (up to 135 bytes), one record corresponding to one horizontal dot row of the plot. The file has record attributes "No Carriage Control." The last three bytes in each record are control-E (hexadecimal 05: plot mode specifier), carriage-return, line-feed. The remaining bytes each use the 6 lower-order bits to represent 6 dots, with the least significant bit representing the left-most dot; 1 implies the corresponding dot is to be filled in. The top two bits are always 1. Thus the maximum number of dots per line is $132 \times 6 = 792$, corresponding to 13.2 inches at 60 dots per inch. If a plot covers more than one page, no form-feed codes are inserted; rather a sufficient number of blank plot lines are inserted to advance to the top of the next page (792 lines per 11-inch page). If these files are to be printed on a Printronix printer using the standard VAX/VMS line-printer driver (LPDRIVER or LCDRIVER), the system manager must set the characteristics of the printer and associated queue correctly. It is important to ensure that (1) all 8-bit characters, including non-printable control characters, are passed to the printer, (2) lines of 135 bytes or less are not truncated, and (3) no extra formatting commands (e.g., form-feeds) are sent to the printer. The following setup is used on Phobos:

```
$ SET PRINTER/LOWER/CR/PRINTALL LPA0:
$ SET DEVICE/SPOOLED=(SYS$PRINT,SYS2:)  LPA0:
$ DEFINE/FORM/NOTRUNC/NOWRAP DEFAULT 0
$ INITIALIZE/QUEUE/START/ON=LPA0: SYS$PRINT -
     /DEFAULT=(FLAG=ONE)/PROT=(W:RW)/SCHED=NOSIZE
```

Obtaining hardcopy: On VAX/VMS machines, Printronix plot files can be printed with a standard `PRINT` command, but the `/PASSALL` qualifier *must* be included. (If the printer is setup as in the example above, `/NOFEED` can be substituted for `/PASSALL`, but this is not recommended.) Examples:

```
$ PRINT/NOTIFY/PASSALL PGPLOT.PRPLOT
```

On Phobos:

```
$ PRINT/PASSALL PGPLOT.PRPLOT
```

On Deimos:

```
$ PPRINT/PASSALL PGPLOT.PRPLOT
```

On Starlink:

```
$ PRINT/PASSALL/QUEUE=SYS_PRINTRONIX PGPLOT.PRPLOT
```

On Ikaros (Convex-Unix):

```
lpr -l PGPLOT.PRPLOT
```

D.6 VT125 (DEC REGIS terminals)

Supported device: Digital Equipment Corporation VT125, VT240, or VT241 terminal; other REGIS devices may also work.

Device type code: `/VT125`.

Default file name (VMS): `TT:PGPLOT.VTPLOT`. This usually means the terminal you are logged in to (logical name `TT`), but the plot can be sent to another terminal by giving the device name, *e.g.*, `TTC0:/VT`, or it can be saved in a file by specifying a file name, *e.g.*, `CITSCR:[TJP]XPLOT/VT` (in this case a disk name must be included as part of the file name).

Default file name (Unix): `/dev/tty`, the terminal you are logged in to. the plot can be sent to another terminal by giving the device name, or it can be saved in a file by specifying a file name.

Default view surface dimensions: Depends on monitor.

Resolution: The default view surface is 768 (horizontal)$\times$ 460 (vertical) pixels. On most Regis devices, the resolution is degraded in the vertical direction giving only 230 distinguishable raster lines. (There are actually 240 raster lines, but 10 are reserved for a line of text.)

Color capability: Color indices 0–3 are supported. By default, color index 0 is black (the background color). Color indices 1–3 are white, red, and green on color monitors, or white, dark grey, and light grey on monochrome monitors. The color representation of all the color indices can be changed, although only a finite number of different colors can be obtained (see the manual for the terminal).

Input capability: The graphics cursor is a blinking diamond-crosshair. The user positions the cursor using the arrow keys and PF1–PF4 keys on his keyboard [Note: NOT the keyboard of the terminal on which he is plotting, if that is different.] The arrow keys move the cursor in the appropriate direction; the size of the step for each keystroke is controlled by the PF1–PF4 keys: PF1 $\Rightarrow$ 1 pixel, PF2 $\Rightarrow$ 4 pixels, PF3 $\Rightarrow$ 16 pixels, PF4 $\Rightarrow$ 64 pixels. [The VT240 terminal has a built-in capability to position the cursor, but PGPLOT does not use this as it is not available on the VT125.] The user indicates that the cursor has been positioned by typing any character other than an arrow or PF1-PF4 key [control characters, *e.g.*, `^C`, and other special characters should be avoided, as they may be intercepted by the operating system].

File format: A REGIS plot file is formatted in records of 80 characters or less, and has no carriage-control attributes. The records are grouped into "buffers," each of which begins with ⟨esc⟩`Pp` to put the terminal into graphics mode and ends with ⟨esc⟩\ to put it back into text mode. The terminal is in graphics mode only while a buffer is being transmitted, so a user's program can write to the terminal at any time (in text mode) without worrying if it might be in graphics mode. Everything between the escape sequences is REGIS: see the VT125 or VT240 manual for an explanation. PGPLOT attempts to minimize the number of characters in the REGIS commands, but REGIS is not a very efficient format. It does have the great advantage, though, that it can easily be examined with an editor. The file may also contain characters outside the ⟨esc⟩`Pp`... ⟨esc⟩\ delimiters, *e.g.*, escape sequences to erase the text screen and home the cursor.

The following escape sequences are used:

⟨esc⟩`[2J`	– Erase entire screen (text)
⟨esc⟩`[H`	– Move cursor to home position
⟨esc⟩`Pp`	– Enter REGIS graphics mode
⟨esc⟩\	– Leave REGIS graphics mode

PGPLOT uses a very limited subset of the REGIS commands supported by the VT125 and VT240. The following list summarizes the REGIS commands presently used.

Initialization: the following standard commands are used to initialize the device every time a new frame is started; most of these restore a VT125 or VT240 to its default state, but the screen addressing mode is nonstandard.

`;`	– resynchronize
`W(R)`	– replace mode writing
`W(I3)`	– color index 1
`W(F3)`	– both bit planes
`W(M1)`	– unit multiplier
`W(N0)`	– negative off
`W(P1)`	– pattern 1
`W(P(M2))`	– pattern multiplier 2
`W(S0)`	– shading off
`S(G1)`	– select graphics plane [Rainbow REGIS]
`S(A[0,479][767,0])`	– screen addressing, origin at bottom left
`S(I0)`	– background dark
`S(S1)`	– scale 1
`S(M0(L0)(AL0))`	– output map section 0 (black)
`S(M1(L30)(AH120L50S100))`	– output map section 1 (red/dim grey)
`S(M2(L59)(AH240L50S100))`	– output map section 2 (green/light grey)
`S(M3(L100)(AL100))`	– output map section 3 (white)

Drawing lines: the `P` and `V` commands are used with absolute coordinates, relative coordinates, and pixel vectors. The `(B)`, `(S)`, `(E)`, and `(W)` modifiers are not used. Coordinates which do not change are omitted.

`P[x,y]` – move to position, *e.g.* `P[499,0]`
`V[x,y]` – draw vector to position, *e.g.* `V[][767][,479][0][,0]`

Line attributes: the line style and line color attributes are specified with `W` commands, *e.g.*

`W(P2)` – line style 2
`W(I2)` – intensity (color index) 2

and `S` commands are used to change the output map. The PGPLOT color indices 0, 1, 2, 3 correspond to output map sections 0, 3, 1, 2.

Obtaining hardcopy: A hardcopy of the plot can be obtained using a printer attached to the VT125/VT240 terminal (see the instruction manual for the terminal). A plot stored in disk file can be displayed by copying it to a suitable terminal; *e.g.*, use `TYPE` on VMS or `cat` on Unix.

D.7 VAX Workstations

Driver: WSDRIVER, version 4.1 (1989 Jun 7), by S. C. Allendorf.

Supported device: This driver should work with all VAX/VMS workstations running VWS software; it requires the UISSHR shareable image provided by DEC.

Device type code: `/WS`.

Default device name: `PGPLOT`. Output is always directed to device `SYS$WORKSTATION`; the "device name" provided by the user is used to label the PGPLOT window.

Default view surface dimensions: Depends on monitor.

Resolution: Depends on monitor.

Color capability: VAX workstations have 1, 4, or 8 bitplanes. On 1-plane devices, there are only two colors (background = white, color index 1 = black). On 4-plane devices, color indices 0–11 are available (4 indices are reserved for text windows and pointers). On 8-plane systems, color indices 0–249 are available (6 indices are reserved for text windows and pointers).

Input capability: The cursor is controlled by the mouse or the keypad (arrow keys and PF1–PF4) available on the controlling (DEC-like) keyboard. The user positions the cursor, and then types any key on the controlling keyboard. The mouse buttons are not used.

Notes: The displayed window is deleted when PGEND is executed or on program exit. PGPLOT requests confirmation from the user before deleting the window. Type a carriage-return at the prompt when you are ready to continue. This makes it impossible to overlay a plot created by one program on a plot created by another. (The `/APPEND` qualifier which allows this for other devices has no effect on device `/WS`.) PGPLOT uses a window which is nominally 11 inches wide by 8.5 inches tall, i.e., the same size as you would get in a hardcopy. If you prefer a vertical orientation, execute the following command before running the program:

```
$ DEFINE PGPLOT_WS_ASPECT PORTRAIT
```

Substitute `LANDSCAPE` for `PORTRAIT` to revert to horizontal orientation.

VAXstations can also be used in Tektronix emulation mode. If you run a process in a Tektronix emulation window, you can use device specification `/TEK` to tell PGPLOT to plot in Tektronix mode within the same window. If you run in a VT220 window, you can tell PGPLOT to create a new Tektronix window and plot in it by giving a device specification `TK:/TEK`. (`TK:` is the VMS device name of the Tektronix emulator.) This has one problem: the window will be deleted as soon as your program calls PGEND or exits; you may need to add a user-prompt in your program before the call of PGEND.

D.8 Sun Workstations

Driver: SVDRIV, by Brian M. Sutin (sutin@astro.umd.edu), 1989 May 19.

Supported device: This driver should work with all Sun workstations running the SunView environment.

Device type code: `/SUNVIEW`.

Default device name: none. Output is always directed to the workstation screen.

Default view surface dimensions: Depends on monitor.

Resolution: PGPLOT uses a square window with 500×500 pixels.

Color capability: 32 colors, 16 pre-defined; white background.

Input capability: The cursor is controlled by the mouse. The user positions the cursor, and then types any key on the controlling keyboard. The mouse buttons can be used instead of keystrokes.

Notes: The displayed window is deleted when PGEND is executed or on program exit. PGPLOT requests confirmation from the user before deleting the window.

D.9 Grinnell

Supported device: Grinnell GMR-270 Image Display System.

Device type code: `/GRinnell`.

Default device name: `TV_DEVICE` (a logical name, usually defined by the system manager).

Default view surface dimensions: Depends on monitor.

Resolution: The full view surface is 512×512 pixels.

Color capability: Color indices 0–255 are supported. The default color representation is as listed in Chapter 5. The representation of all color indices can be changed.

Input capability: The graphics cursor is a white cross-hair. The user positions the cursor using the arrow keys and PF1–PF4 keys on his terminal keyboard (`SYS$COMMAND`). The arrow keys move the cursor in the appropriate direction; the size of the step for each keystroke is controlled by the PF1–PF4 keys: PF1 $\Rightarrow$ 1 pixel, PF2 $\Rightarrow$ 4 pixels, PF3 $\Rightarrow$ 16 pixels, PF4 $\Rightarrow$ 64 pixels. The user indicates that the cursor has been positioned by typing any character other than an arrow or PF1-PF4 key [control characters, *e.g.*, `^C`, and other special characters should be avoided, as they may be intercepted by the operating system].

File format: It is not possible to send Grinnell plots to a disk file.

Obtaining hardcopy: Not possible.

D.10 IVAS

Supported device: International Imaging Systems IVAS Display Processor.

Device type code: `/IVAS`.

Default device name: `/dev/ga0` (Unix).

Default view surface dimensions: Depends on monitor.

Resolution: The full view surface is 1024×1024 pixels.

Color capability: Color indices 0–15 are supported. The default color representation is as listed in Chapter 5. The representation of all color indices can be changed.

Input capability: The graphics cursor is a yellow cross. The user positions the cursor using the mouse, and indicates that the cursor has been positioned by typing any character other than an arrow or PF1-PF4 key [control characters, *e.g.*, `^C`, and other special characters should be avoided, as they may be intercepted by the operating system].

File format: It is not possible to send IVAS plots to a disk file.

Obtaining hardcopy: Not possible.

D.11 Sigma ARGS

Supported device: Sigma ARGS color graphic display.

Device type code: `/ARgs`.

Default device name: `ARGS_DEVICE` (a logical name).

Default view surface dimensions: Depends on monitor.

Resolution: The full view surface is 512×512 pixels.

Color capability: Color indices 0–255 are supported. The default color representation is as listed in Chapter 5. The representation of all color indices can be changed.

Input capability: maybe....

File format: It is not possible to send ARGS plots to a disk file.

Obtaining hardcopy: Not possible.

D.12 Tektronix 4006, 4010

Supported device: Tektronix 4006 and 4010 Series Storage Tube terminals, and "emulators."

Device type code: `/TEk4010`.

Default device name: `TT` (a logical name, usually equivalent to the logged-in terminal).

Default view surface dimensions: Depends on monitor.

Resolution: The full view surface is nominally 1024 (horizontal)$\times$ 768 (vertical) pixels, but the actual resolution varies from device to device.

Color capability: None. Only color index 1 is permitted, and requests for other color indices are ignored. It is not possible to change color representation, or to erase by using color index 0.

Input capability: Maybe....

File format: It is not possible to send Tektronix plots to a disk file.

Obtaining hardcopy: A hardcopy of the plot can be obtained using a Tektronix hardcopy unit attached to the terminal.

D.13 Tektronix 4100

Supported device: Tektronix 4100-series terminals.

Device type code: `/TK4100`.

Default device name: `TT` (a logical name, usually equivalent to the logged-in terminal).

Default view surface dimensions: Depends on monitor.

Resolution: The view surface is nominally 4096 (horizontal) $\times$ 3072 (vertical) pixels, but the true resolution depends on which terminal is used, and is usually much less than this.

Color capability: Color indices 0–16 are supported. The default color representation is as listed in Chapter 5. The representation of all color indices can be changed.

Input capability: Not yet implemented.

File format: It is not possible to send Tektronix plots to a disk file.

Obtaining hardcopy: It is possible to obtain a hardcopy of the plot (in color, even) using a printer attached to the terminal.

D.14 Retrographics

Supported device: Digital Engineering, Inc., Retrographics modified VT100 terminal (VT640).

Device type code: `/REtro`.

Default device name: `TT:` (the logged-in terminal).

Default view surface dimensions: Depends on monitor.

Resolution: The full view surface is 1024 (horizontal)$\times$ 780 pixels.

Color capability: Color indices 0 (erase, black) and 1 (bright: usually green) are supported. It is not possible to change color representation.

Input capability: The graphics cursor is a crosshair across the entire screen. The user positions the cursor using the four arrow keys on the keyboard of the Retrographics terminal. "By striking the desired directional arrow key, the crosshair will move across the display screen at the rate of one dot per keystroke. Applying a constant pressure on an arrow key will cause the crosshair to move at a continuous rapid rate; releasing the key will stop the crosshair's movement." The user indicates that the cursor has been positioned by typing any printable ASCII character on the keyboard of the Retrographics terminal. Most control characters (*e.g.*, `^C`) are intercepted by the operating system and cannot be used.

File format: It is not possible to send Retro plots to a disk file.

Obtaining hardcopy: Not possible.

D.15 Null Device

Supported device: The "null" device can be used to suppress all graphic output from a program.

Device type code: `/Null`.

Default device name: None (the device name, if specified, is ignored).

Default view surface dimensions: Undefined.

Resolution: Undefined.

Color capability: Color indices 0–255 are accepted.

Input capability: None.

File format: None.

Obtaining hardcopy: Not possible.

D.16 Canon

Supported device: Canon LBP-8/A2 Laser printer. Conforms to ISO 646, 2022, 2375 and 6429 specifications. VDM (graphics) conforms to proposed American National Standard VDM mode.

Device type code: `/CAnon` (landscape mode), `/VCanon` (portrait mode) and `/BCanon` (bitmap in landscape mode only).

Default file name: `PGPLOT.CAN`

Default view surface dimensions: 24 cm by 19 cm.

Resolution: 300 pixels per inch in both directions.

Color capability: Color indices 0 (erase) and 1 (black) are supported. Note, hardware polygon fill is used and colors 0-11 control the fill pattern.

Input capability: None.

File format: Variable length records with Carriage control of LIST.

Obtaining hardcopy: If printer is connected to a terminal line (RS-232 option) then printing the file on the corresponding queue should suffice. If the printer is connected using the Centronics interface that appears the to VAX as an LP device then it is important to ensure that (1) all 8 bit characters are passed to the printer (2) lines longer than 132 bytes are not truncated, and (3) no extra formatting commands (e.g. form-feeds) are sent to the printer. This can be done with the VMS command:

```
$ SET PRINT/PASSALL/LOWER/CR <device>
```

Note, some interface boards have a option to append a carriage return after a formfeed or LF character, it is necessary that this be disabled. The file should be printed with the /PASSALL qualifier i.e.,

```
$ PRINT/PASSALL <filename>
```

Note, SET PRINT/PASSALL and PRINT/PASSALL do not do the same things and hence PASSALL is required in both locations.

Note: The BCDRIVER produces a bitmap that then can be printed on the Canon. The default size is 1556 blocks and takes 5 min (parallel) or 15 min (serial 9600 baud) to print. Thus for simple line graphs CADRIVER produces much smaller files (typically <100 blocks) that that plot in <30 sec. However, for complex graphs, for example those obtained with PGGRAY, BCDRIVER will produce the smaller file and plot faster. Therefore, it is suggested that sites with Canon laser printers should support both drivers.

D.17 Colorwriter 6320 Plotter

Supported device: Gould (now Bryans) Colourwriter 6320 or any device obeying Gould Plotter Language.

Device type code: `/CW6320`

Default device name: `$PLOTTER1` (Defined system logical name)

Default view surface dimensions: 280mm by 360mm (A3)

Resolution: 0.025mm

Colour Capability: Up to 10 pens.Default is pen 1 which is picked up on initialization without a call to PGSCI.Calls to PGSCI are interpreted as the pen number and colours therefore depend on how the pens have been loaded into the stalls.If a call is made for a pen higher than 10 the selected pen defaults to 1.

Input Capability: Possible but not supported.

File format: Ascii character strings.It is possible to send the data to a file which can then be copied to the plotter or examined on a terminal.

Obtaining hard copy: PGPLOT has been fixed to send the plot directly to the plotter without an intermediate file.

D.18 Ikon

Supported device: Digisolve Ikon Pixel Engine

Device type code: `/IKon.`

Default device name: `IKON_DEFAULT` (a logical name).

Default view surface dimensions: Depends on monitor.

Resolution: The full view surface is 1024 by 780 pixels.

Color capability: Color indices 0-255 are supported. The default representation is listed in Chapter 5 of the PGPLOT manual. The representation of all color indices can be changed.

Input capability:

File format: It is not possible to send IKON plots to a disk file.

Obtaining hardcopy: Not possible.

D.19 Zeta

Supported device: Zeta 8 Digital Plotter.

Device type code: `/ZEta`

Default file name: `PGPLOT.ZET`

Default view surface dimensions: 11 inches by 11 inches. Current version does not allow larger plots although the manual indicates plots up to 144 feet are possible.

Resolution: This version is written for the case where the resolution switch is set to .025 mm. Actual resolution depends on thickness of pen tip.

Color capability: Color indices 1 to 8 are supported corresponding to pens 1-8. It is not possible to erase lines.

Input capability: None.

File format: Variable length records with Carriage control of LIST.

Obtaining hardcopy: On Starlink print the file on the queue associated with the Zeta plotter. If the Plotter is attached to a terminal line, then TYPEing the file at the terminal will produce a plot. On Starlink:

```
$ PRINT/NOFEED/QUE=ZETA PGPLOT.ZET
```

To stop a Zeta plot job, once it has been started, use the buttons on the plotter. Press PAUSE, NEXT PLOT and CLEAR. Only after this sequence is it safe to delete the job from the ZETA Queue. Failing to press the NEXT PLOT button will not correctly advance the paper. Failing to press CLEAR but, deleting the current job can prevent the following plot from being plotted.

Appendix E

WRITING A DEVICE HANDLER

E.1 Introduction

PGPLOT can be configured for a particular installation by adding or removing "device handlers". A device handler is a subroutine which handles all the device-specific aspects of graphical output for a particular device or class of devices.

All graphical output is handled by a "device dispatch routine" in PGPLOT, called GREXEC. This routine calls on the appropriate device handler to perform the output. Reconfiguring PGPLOT involves modifying the GREXEC routine to use a different set of device handlers; no other changes to PGPLOT are needed.*

To write a new device handler, it is simplest to start by modifying an existing one. This Appendix explains what the device handler must do, but it does not explain how to do it—which is, of course, very hardware-dependent.

The supported device types fall into three classes, and when adding a new device you should determine which class it belongs to, and then add the new device by copying and modifying the support routines for one of the existing devices in this class. The three classes are:

1. Bit map devices (*e.g.*, Versatec). The complete image is assembled in memory as a bit map and then written in an output file in a format acceptable to the device. The existing devices in this class use one bit per pixel, but more bits could be allocated.
2. Instruction stream devices (*e.g.*, QMS): Graphical instructions are written sequentially in the output file using Fortran I/O. Usually the instructions are composed of printable ASCII characters, but this is not essential.

---

* In the VMS version of PGPLOT, the modification of GREXEC can be done automatically using the command procedure `NEWEXEC.COM` which finds all the device handlers currently installed in the `[PGPLOT.SOURCE]` directory.

3. Direct I/O devices (*e.g.*, Grinnell): devices that require special commands that cannot be encoded in text strings use the low-level operating system I/O routines (SYS$QIO in VMS) to directly control the device. It is not possible to store images in a disk file for these devices.

E.2 Device handler interface

A device handler is a Fortran subroutine. It is called by the GREXEC device dispatch routine whenever PGPLOT needs to determine device-specific information or perform graphical output. The name of the subroutine must be of the form *xx*DRIV, where *xx* is a two-character code for the device type, usually the first two letters of the type; this code must (of course) be different for each different device handler.

```
SUBROUTINE xxDRIV (OPCODE, RBUF, NBUF, CHR, LCHR)
INTEGER        OPCODE
REAL           RBUF(*)
INTEGER        NBUF
CHARACTER*(*)  CHR
INTEGER        LCHR
```

The first argument (OPCODE) is an integer "operation code" which specifies what operation the device handler is to perform; it is an input parameter to the subroutine (see Table E.1). The other arguments are used for both input and output, and their meaning depends on the value of the operation code. Not all arguments are used for every operation code. RBUF is a floating-point array used to pass numerical data to or from the device handler, and NBUF indicates how many elements of the array are used. CHR is a character variable used to pass character data to or from the device handler, and LCHR indicates how many characters are used. NBUF or LCHR should be set to zero if no data of the corresponding type are passed. If the function requested by the operation code (OPCODE) is not implemented in the device handler, the subroutine should set NBUF = -1 before returning.

The device handler subroutine can communicate with PGPLOT *only* through the arguments. It should not attempt to reference the PGPLOT common blocks (this is because the internal structure of the PGPLOT common blocks may change).

Table E.1 Device Handler Operation Codes

Opcode	*Function*
1	Return device name
2	Return maximum dimensions of view surface, and range of color index
3	Return device scale
4	Return device capabilities
5	Return default device/file name
6	Return default size of view surface
7	Return miscellaneous defaults
8	Select device
9	Open workstation
10	Close workstation
11	Begin picture
12	Draw line
13	Draw dot
14	End picture
15	Set color index
16	Flush buffer
17	Read cursor
18	Erase alpha screen
19	Set line style
20	Polygon fill
21	Set color representation
22	Set line width
23	Escape function
24	Rectangle fill
25	Set fill pattern
26	Line of pixels

E.3 Handler state

PGPLOT will send commands to the device handler in a set sequence. Inquiry commands (opcodes 1–7) may be sent at any time, whether or not a device has been selected for output. The *open workstation* and *close workstation* commands are used to open and close a device. The *begin picture* and *end picture* commands are used to start and finish a "frame" (one page on a hardcopy device). Graphical output commands (opcodes 12–13, 16–23) are only used between *begin picture* and *end picture*. Thus the sequence of commands for a plot consisting of two frames will be:

```
open workstation
    begin picture
        (graphical output commands)
    end picture
    begin picture
        (graphical output commands)
    end picture
close workstation
```

Any violation of this sequence is due to a bug in PGPLOT.

E.4 Summary of operations

OPCODE = 1, Return device name. This is an inquiry function; the handler returns the name by which the the user will refer to the device type, *e.g.*, 'PRINTRONIX' for a Printronix device handler. This name must be different for each device handler installed in PGPLOT, and should preferably be unique in the first two or three characters.

CHR(:LCHR) (*returned*): the device type supported by the handler.

OPCODE = 2, Return maximum dimensions of view surface, and range of color index. This is an inquiry function; the handler returns the maximum dimensions of the plot surface, and the range of color indices available. On interactive devices, these will usually be the same as the default dimensions. On hardcopy devices which plot on roll or fanfold paper, the maximum dimensions may be larger. All dimensions are in device coordinates. All devices should support color indices 0 and 1; color and gray-scale devices will allow color indices > 1 up to a device-dependent maximum value (which should not exceed 255). Color index 0 is the background color and is used to erase; if it is not possible to erase by overwriting in the background color, then requests to write in color index 0 should be ignored.

RBUF(1) (*returned*): Minimum physical x value (set to zero).

RBUF(2) (*returned*): Maximum physical x value (a value of -1 indicates no effective maximum).

RBUF(3) (*returned*): Minimum physical y value (set to zero).

RBUF(4) (*returned*): Maximum physical y value (a value of -1 indicates no effective maximum).

RBUF(5) (*returned*): Minimum allowed color index (usually 0).

RBUF(6) (*returned*): Maximum allowed color index (in range 1–255).

OPCODE = 3, Return device scale. This is an inquiry function; the handler returns the device scale in device coordinate units per inch. Usually, the units of the device coordinates are pixels, so this also gives the physical resolution in pixels per inch. For hardcopy devices, the values should be as accurate as possible, to ensure that an image has the correct scale. For video display terminals and other devices where the scale is variable, nominal values should be returned.

RBUF(1) (*returned*): x scale in device coordinates per inch.

RBUF(2) (*returned*): y scale in device coordinates per inch.

RBUF(3) (*returned*): "pen diameter" in device coordinates (i.e., the width of a hardware line); this value is used by PGPLOT when emulating thick lines and polygon fill.

OPCODE = 4, Return device capabilities. This is an inquiry function which is used to inform PGPLOT of the device's capabilities. If the device lacks a capability in hardware, PGPLOT will try to emulate it.

CHR(1:10) (*returned*): each character in this string defines whether a capability exists:

CHR(1:1) = 'H' if the device is a hardcopy device, 'I' if it is an interactive device. On an interactive device, the image is visible as it is being drawn, while on a hardcopy device it cannot be viewed until the workstation is closed.

CHR(2:2) = 'C' if a cursor is available, 'N' if not. PGPLOT cannot emulate a cursor if none is available.

CHR(3:3) = 'D' if the hardware can draw dashed lines, 'N' if it cannot. PGPLOT emulates dashed lines by drawing line segments. Software emulation is usually superior to hardware dashed lines, and not much slower, so CHR(3:3) = 'N' is recommended.

CHR(4:4) = 'A' if the hardware can fill arbitrary polygons with solid color, 'N' if it cannot. PGPLOT emulates polygon fill by drawing horizontal or vertical lines spaced by the pen diameter (see OPCODE = 3).

CHR(5:5) = 'T' if the hardware can draw lines of variable width, 'N' if it cannot. PGPLOT emulates thick lines by drawing multiple strokes spaced by the pen diameter. Note that thick lines are supposed to have rounded ends, as if they had been drawn by a circular nib of the specified diameter.

CHR(6:6) = 'R' if the hardware can fill rectangles with solid color, 'N' if it cannot. If this feature is not available, PGPLOT will treat the rectangle as an arbitrary polygon. In this context, a 'rectangle' is assumed to have its edges parallel to the device-coordinate axes.

CHR(7:7) = 'P' if the driver understands the pixel primitives (opcode 26), 'N' otherwise.

CHR(8:10) : reserved for future use; the device handler should return 'N' in all these positions.

OPCODE = 5, Return default device/file name. This is an inquiry routine. The device handler returns the device or file name to be used if the PGPLOT device specification does not include one. (On VMS, the default file name is also used to fill in missing fields of the supplied file name, e.g., disk, directory, and file type.)

CHR(:LCHR) (*returned*): default device/file name.

OPCODE = 6, Return default size of view surface. This is an inquiry function; the handler returns the default dimensions of the plot surface in device coordinates. At present, PGPLOT assumes that the device coordinates of the bottom left corner are $(0,0)$.

RBUF(1) (*returned*): default x-coordinate of bottom left corner (must be zero).

RBUF(2) (*returned*): default x-coordinate of top right corner.

RBUF(3) (*returned*): default y-coordinate of bottom left corner (must be zero).

RBUF(4) (*returned*): default y-coordinate of top right corner.

OPCODE = 7, Return miscellaneous defaults. This is an inquiry routine. The handler returns a scale-factor to be used for the "obsolete character set" used by old GRPCKG routines but not by PGPLOT.

RBUF(1) (*returned*): character scale factor (integer, ≥ 1).

OPCODE = 8, Select device. This opcode is reserved for future use. At present, each device handler can handle only one open device at once. Future versions of PGPLOT will allow more than one device to be open at once, and this opcode will be used to select the active device.

RBUF(1) (*input*): plot ID.

RBUF(2) (*input*): unit or channel number of selected device (as returned by *open workstation*.

OPCODE = 9, Open workstation. Allocate an I/O channel to the requested device and open the device. Any hardware resets that need to be done once for a plot session (which could consist of several frames) should be done here. Allocate buffer, if its size is fixed for the device. No visible I/O should be performed on an interactive device: e.g., the screen should not be cleared; this should be deferred until the *begin picture* call.

RBUF(1) (*returned*): identification number of the I/O channel opened; PGPLOT will use this number in subsequent *select device* calls for this device (see OPCODE = 8).

RBUF(2) (*returned*): error flag; 1.0 indicates that the workstation was opened successfully; any other number indicates an error.

RBUF(3) (*input*): if $\neq 0$, the device specification included the /APPEND flag. If this flag is specified, the device handler should suppress any initial screen erase so that the new image is superimposed on any previously displayed image. The device handler may ignore this if it is inappropriate (e.g., for a hardcopy device).

CHR(:LCHR) (*input*): the file/device to be opened. On VMS, this will be a physical device name, an RMS file name, or a logical name.

OPCODE = 10, Close workstation. Close the device opened by the *open workstation* command, and deallocate any resources allocated for the device (e.g., memory, I/O channels).

OPCODE = 11, Begin picture. Prepare the workstation for plotting. This command has two arguments which specify a size for the view surface overriding the default size; if the device handler is unable to change the size of the view surface, it may ignore these arguments. On interactive devices, erase the screen.

RBUF(1) (*input*): maximum x coordinate.

RBUF(2) (*input*): maximum y coordinate.

OPCODE = 12, Draw line. Draw a straight line from device coordinates (x_1, y_1) to (x_2, y_2) using the current line attributes (color index, line style, and line width). The coordinates are floating point, and may need to be rounded to the nearest integer before they are passed to the hardware.

RBUF(1) (*input*): x_1.

RBUF(2) (*input*): y_1.

RBUF(3) (*input*): x_2.

RBUF(4) (*input*): y_2.

OPCODE = 13, Draw dot. Draw a dot at device coordinates (x, y) using the current line attributes (color index and line width). The result should be an approximation to a filled circle of diameter equal to the line width, or a dot of minimum size if line width is 0. The coordinates are floating point, and may need to be rounded to the nearest integer before they are passed to the hardware.

RBUF(1) (*input*): x.

RBUF(2) (*input*): y.

OPCODE = 14, End picture. Terminate the current frame. On hardcopy devices always advance the paper. On interactive devices, clear the screen only if requested. Deallocate buffers that were created by *begin picture* (OPCODE = 11).

RBUF(1) (*input*): if $\neq 0$, clear screen.

OPCODE = 15, Set color index. Set the color index for subsequent plotting. The default color index is 1.

RBUF(1) (*input*): color index; in range defined by OPCODE = 2.

OPCODE = 16, Flush buffer. If the handler is buffering output to an interactive device, it should flush its buffers to ensure that the displayed image is up to date. Hardcopy devices can ignore this opcode.

OPCODE = 17, Read cursor. This function is not used if OPCODE = 4 indicates that the device has no cursor. The handler should make the cursor visible at position (x, y), allow the user to move the cursor, and wait for a key stroke. It should then return the new cursor (x, y) position and the character (key stroke) typed. (If it is not possible to make the cursor visible at a particular position, the handler may ignore the requested (x, y) coordinates.)

RBUF(1) (*input/returned*): x position of cursor.

RBUF(2) (*input/returned*): y position of cursor.

CHR(:1) (*returned*): character typed by user.

OPCODE = 18, Erase alpha screen. If graphics device is a terminal that can display both graphics and text on the same screen, clear the text screen, leaving graphics unchanged. All other devices should ignore this opcode.

OPCODE = 19, Set line style. This opcode is not used if OPCODE = 4 indicates that the device does not support hardware dashing; PGPLOT will use software-generated dashed lines.

RBUF(1) (*input*): requested line style (integer 1–5).

OPCODE = 20, Polygon fill. This function is not used if OPCODE = 4 indicates that the device does not support hardware polygon fill. The polygon may be arbitrarily complex (concave or re-entrant); if the hardware cannot cope with this, the handler should set the OPCODE = 4 response to disable hardware fill. If hardware fill is enabled, the handler should respond to this function by filling the polygon with the current color index. To draw an N-sided polygon, PGPLOT uses this opcode $N + 1$ times.

First call:

RBUF(1) (*input*): number of points N in polygon.

For next N calls:

RBUF(1) (*input*): x value.

RBUF(2) (*input*): y value.

OPCODE = 21, Set color representation. Assign the specified (R, G, B) color, or the best available approximation, to the specified color index. If colors cannot be changed dynamically, ignore the request.

RBUF(1) (*input*): color index (integer, in range defined by OPCODE = 2).

RBUF(2) (*input*): red component (0.0–1.0).

RBUF(3) (*input*): green component (0.0–1.0).

RBUF(4) (*input*): blue component (0.0–1.0).

OPCODE = 22, Set line width. This function is not used if OPCODE = 4 indicates that the device does not support hardware thick lines. Subsequent lines and dots should be drawn with the requested width, or the closest available approximation. The units of line-width are 0.005 inches. A requested line-width of zero should give the narrowest line available on the device ("hair line").

RBUF(1) (*input*): requested line width, in units of 0.005 inch.

OPCODE = 23, Escape function. This function allows an arbitrary character string to be sent to the device handler. The interpretation is up to the handler; usually, the string will be sent directly to the device or ignored. Use of this function should be avoided.

CHR(:LCHR) (*input*): character string.

OPCODE = 24, Rectangle fill. This function is not used if OPCODE = 4 indicates that the device does not support hardware rectangle fill.

RBUF(1), RBUF(2) (*input*): x, y coordinates of lower left corner of rectangle.

RBUF(3), RBUF(4) (*input*): x, y coordinates of upper right corner of rectangle.

OPCODE = 25, Set fill pattern. This function is not yet implemented.

OPCODE = 26, Line of pixels. This function is not used if OPCODE = 4 indicates that the device does not support this function. It is used to write a horizontal line of pixels on the device screen with specified color indices; it should be more efficient to do this with one device driver call rather than separate calls for each pixel. This operation is used for gray-scale and color imaging (e.g., routine PGGRAY). If the device handler implements this operation, it is important that the device coordinates should be true pixel numbers.

RBUF(1), RBUF(2) (*input*): x, y coordinates of the first pixel to be written. These should be integer pixel numbers in the device coordinate system (passed as REAL numbers).

RBUF(3)...RBUF(NBUF) (*input*): color indices for n pixels to be filled in, starting at (x, y) and ending at $(x+n-1, y)$. The values should be valid integer color indices for the device (passed as REAL numbers). The number of pixels is specified by the argument NBUF: n = NBUF − 2.

E.5 Testing a new device handler

Several of the example programs can be used to test that PGPLOT has been installed correctly, or that a new device handler is working. For a complete test, I recommend running the following programs on each installed device type.

PGEX17 This tests most of the features of PGPLOT and will reveal most device-handler errors.

1. It draws a rectangle enclosing the entire view surface. All four sides of this rectangle should be visible. It then draws the bisectors of the sides, which intersect in the center of the rectangle, and a concentric circle. The circle should always be circular, not elliptical, independent of the aspect ratio of the view surface. The dimensions of the rectangle and circle are typed on the terminal before the program exits; these dimensions should be checked against the actual dimensions for hardcopy devices (for TV-type devices the exact dimensions are not important).
2. It draws 5 vertical lines of width 1 in the five different line-styles.
3. It draws 5 horizontal lines of width 1 through 5.
4. It draws 21 dots, using pen-widths 1 through 21. Check particularly that the first (lowest) dot, of diameter 1, is visible.

5. It draws 16 vertical lines using color indices 0–15. The first line, in color index zero, overwrites the vertical bisector of the rectangle. If erase-mode is implemented, this will erase the line drawn in step 1.
6. It outlines and fills four polygons (ducks) in color indices 0–3. The interior of the first polygon should be erased if erase-mode is implemented.
7. Finally it erases (or attempts to erase) a rectangle at the top of the image, and writes the version number of PGPLOT and the name of the device type within the rectangle. If this text does not appear, probably the environment variable `PGPLOT_FONT` is not defined correctly.

PGDEMO1 This draws several simple graphs using the basic PGPLOT routines. The main feature that this program tests that is not tested by PGEX17 is the ability to clear the screen or start a new page. On interactive devices, the program should prompt for a carriage-return before starting the second and subsequent graphs. A large number of the PGPLOT routines are tested by this program.

PGDEMO2 The first image produced by PGDEMO2 is mostly useful for testing TV-type displays which support many color-indices. It draws an alignment grid in two shades of gray, and patches of color using indexes 0–15. Subsequent images demonstrate the graph markers and text fonts.

PGEX15 PGEX15 can be use to test the cursor (routine `PGCURSE`). It draws a frame around the view surface and allows the user to position the cursor. The keystroke code typed by the user and the cursor position are displayed on the terminal. To exit from the program, type a slash (`/`).

Appendix F

CALLING PGPLOT FROM A C PROGRAM

F.1 Introduction

It is possible to call PGPLOT routines from a program written in C. The methods for calling a Fortran subroutine from a C program are operating-system dependent, and indeed it is impossible in some systems. The details are rather complicated, and if there is a demand, we could create a C-callable library that hides these details from the user. Examples of the same program written in Fortran-77, VAX/VMS C, and Convex UNIX C are included below.

F.2 VMS

The following is a prescription for calling PGPLOT subroutines from a C program on a VAX running VMS.

1. All arguments (except arrays and character strings) are passed by address using the `&` operator. As you cannot take the address of a constant, constants must be passed in dummy variables.
2. `INTEGER` arguments correspond to C type `long int` or `int`.
3. `REAL` arguments correspond to C type `float`.
4. `CHARACTER` arguments correspond to C character strings, but they must be passed by descriptor. The VAX-C manual explains how to do this using the `$DESCRIPTOR` macro for fixed strings; it is a little more complicated for variable strings.
5. Note that the backslash character (`\`) must be escaped (`\\`).

F.3 Convex UNIX

The following is a prescription for calling PGPLOT subroutines from a C program on the Convex. It may also work on other Berkeley UNIX systems.

1. The main program must be called `MAIN__()` instead of `main()`. I don't know why this is so, but it must have to do with the initialization of the Fortran library.
2. Use the C compiler to compile the program, but use `fc` to load it. This ensures that the Fortran system libraries are scanned.
3. All PGPLOT subroutine names must be typed in lower case, with an underscore appended, e.g., `pgbegin_()`, `pgend_()`.
4. All arguments (except arrays and character strings) are passed by address using the `&` operator. As you cannot take the address of a constant, constants must be passed in dummy variables.
5. `INTEGER` arguments correspond to C type `long int`.
6. `REAL` arguments correspond to C type `float`.
7. For `CHARACTER` arguments, pass a pointer to a C character string, and add the length of the string (number of characters) as an extra `long int` argument at the end of the argument list.
8. Note that the backslash character (`\`) must be escaped (`\\`).

   Example:

   ```
   cc -c example1.c
   fc -o example1 example1.o -lpgplot
   example1
   ```

```
C -----------------------------------------------------------------------
C Demonstration program for PGPLOT (Fortran version).
C -----------------------------------------------------------------------
C
      PROGRAM PGDEMO
C
      INTEGER I
      REAL XS(5),YS(5), XR(100), YR(100)
      DATA XS/1.,2.,3.,4.,5./
      DATA YS/1.,4.,9.,16.,25./
C
C Call PGBEGIN to initiate PGPLOT and open the output device; PGBEGIN
C will prompt the user to supply the device name and type.
C
      CALL PGBEGIN(0,'?',1,1)
C
C Call PGENV to specify the range of the axes and to draw a box, and
C PGLABEL to label it. The x-axis runs from 0 to 10, and y from 0 to 20.
C
      CALL PGENV(0.,10.,0.,20.,0,1)
      CALL PGLABEL('(x)', '(y)', 'PGPLOT Example 1 - y = x\u2')
C
C Mark five points (coordinates in arrays XS and YS), using symbol
C number 9.
C
      CALL PGPOINT(5,XS,YS,9)
C
C Compute the function at 60 points, and use PGLINE to draw it.
C
      DO 10 I=1,60
          XR(I) = 0.1*I
          YR(I) = XR(I)**2
   10 CONTINUE
      CALL PGLINE(60,XR,YR)
C
C Finally, call PGEND to terminate things properly.
C
      CALL PGEND
C
      END
```

```
/* ---------------------------------------------------------------------
 * Demonstration program for PGPLOT called from C [VMS].
 *---------------------------------------------------------------------
 */
#include descrip
main()
{
    int i;
    static float xs[] = {1.0, 2.0, 3.0, 4.0, 5.0 };
    static float ys[] = {1.0, 4.0, 9.0, 16.0, 25.0 };
    float xr[100], yr[100];
    long dummy, nx, ny;
    float xmin, xmax, ymin, ymax;
    long just, axis, n, symbol;
    $DESCRIPTOR(device, "?");
    $DESCRIPTOR(xlabel, "(x)");
    $DESCRIPTOR(ylabel, "(y)");
    $DESCRIPTOR(toplabel, "PGPLOT Example 1 - y = x\\u2");
/*
 * Call PGBEGIN to initiate PGPLOT and open the output device; PGBEGIN
 * will prompt the user to supply the device name and type.
 */
    dummy = 0;
    nx = 1;
    ny = 1;
    pgbegin(&dummy, &device, &nx, &ny);
/*
 * Call PGENV to specify the range of the axes and to draw a box, and
 * PGLABEL to label it. The x-axis runs from 0 to 10, and y from 0 to 20.
 */
    xmin = 0.0;
    xmax = 10.0;
    ymin = 0.0;
    ymax = 20.0;
    just = 0;
    axis = 1;
    pgenv(&xmin, &xmax, &ymin, &ymax, &just, &axis);
    pglabel(&xlabel, &ylabel, &toplabel);
/*
 * Mark five points (coordinates in arrays XS and YS), using symbol
 * number 9.
 */
    n = 5;
    symbol = 9;
    pgpoint(&n, xs, ys, &symbol);
/*
 * Compute the function at 60 points, and use PGLINE to draw it.
 */
    n = 60;
    for (i=0; i<n; i++)
        {
        xr[i] = 0.1*i;
        yr[i] = xr[i]*xr[i];
        }
    pgline(&n, xr, yr);
/*
 * Finally, call PGEND to terminate things properly.
 */
    pgend();
}
```

```
/* ---------------------------------------------------------------------
 * Demonstration program for PGPLOT called from C [Convex UNIX].
 *----------------------------------------------------------------------
 */
MAIN__()
{
    int i;
    static float xs[] = {1.0, 2.0, 3.0, 4.0, 5.0 };
    static float ys[] = {1.0, 4.0, 9.0, 16.0, 25.0 };
    float xr[100], yr[100];
    long dummy, nx, ny, just, axis, n, symbol;
    float xmin, xmax, ymin, ymax;
/*
 * Call PGBEGIN to initiate PGPLOT and open the output device; PGBEGIN
 * will prompt the user to supply the device name and type.
 */
    dummy = 0;
    nx = 1;
    ny = 1;
    pgbegin_(&dummy, "?", &nx, &ny, 1L);
/*
 * Call PGENV to specify the range of the axes and to draw a box, and
 * PGLABEL to label it. The x-axis runs from 0 to 10, and y from 0 to 20.
 */
    xmin = 0.0;
    xmax = 10.0;
    ymin = 0.0;
    ymax = 20.0;
    just = 0;
    axis = 1;
    pgenv_(&xmin, &xmax, &ymin, &ymax, &just, &axis);
    pglabel_("(x)", "(y)", "PGPLOT Example 1 - y = x\\u2", 3L, 3L, 27L);
/*
 * Mark five points (coordinates in arrays XS and YS), using symbol
 * number 9.
 */
    n = 5;
    symbol = 9;
    pgpoint_(&n, xs, ys, &symbol);
/*
 * Compute the function at 60 points, and use PGLINE to draw it.
 */
    n = 60;
    for (i=0; i<n; i++)
        {
        xr[i] = 0.1*i;
        yr[i] = xr[i]*xr[i];
        }
    pgline_(&n, xr, yr);
/*
 * Finally, call PGEND to terminate things properly.
 */
    pgend_();
}
```