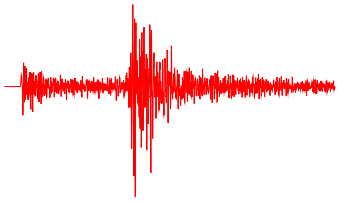
COMPUTER PROGRAMS IN SEISMOLOGY



CALPLOT GRAPHICS

Robert B. Herrmann
Editor

Professor of Geophysics
Department of Earth and Atmospheric Sciences
Saint Louis University

Version 3.30
2004

Contents

PREFACE

Widely available computer graphics are a relatively recent development. Even in 1970, the available devices were few. CALCOMP mechanical plotters were a *de factor* standard for quality, hardcopy output. The Tektronix 401X series set the standard for interactive terminal graphics devices. Because of the desire for a level of programming compatibility, software emulators were developed to mimic the hardware of other manufacturers. Thus Tektronix developed a PLOT-10 software library to enable their terminals to emulate basic CALCOMP calls. Other manufacturers, such as Hewlett-Packard provided similar libraries to implement an industry standard Since then, other standards have been proposed: GKS, Core, Phigs, etc., many of which have since disappeared.

As part of the UNIX V7 system (1979), a set of graphic primitives was provided, together with device drivers for a small number of graphics devices. The UNIX V7 *plot(3)* library and *plot(1)* filters enabled a C program to generate a binary *plot* file which consisted of simple pen up and pen down commands. The interesting aspect of this concept, was that the binary *plot* could be implemented in a manner that was independent of the internal method of number storage, e.g., little endian versus big endian. This meant that the *plot* file could be used machines with SPARC or Intel Architectures without change. A problem with the UNIX V7 *plot* routines was that they only had a C interface and no FORTRAN interface. Another difficulty was that device drivers that used the binary *plot* file output were many because of many different pieces of hardware developed.

The **CALPLOT** package presented here is a merge of these two developments to create a device independent high level graphics language that is upwardly compatible with the earlier industry standard plotting packages that emulated CALCOMP plotters. The **CALPLOT** package so developed turned out to be very robust, so that it has now evolved beyond the original task of emulating a CALCOMP plotter. This evolution was directed by graphics output needs as well as the ability to develop plot drivers that use the hardware capabilities. Thus primitives for shaded area fill, seismic trace shading, different fonts, and line width have been added to the original capabilities. User programs are compiled and, in general, create a device independent graphics metafile that is upwardly compatible with the UNIX V7 metafile. This metafile is then used as input by a device specific plot filter which actually generates the graphic output. A design decision was made that all hard copy be able to be overlain irrespective of the specific hardware.

Since the initial development of CALPLOT in 1983, graphic standards have become accepted. For paper copy, the Adobe PostScript format is accepted. Workstation graphics is build upon X11 for the UNIX/LINUX systems and Windows calls for MS Windows.

Supported free software permits converting PostScript to any number of raster graphics formats (JPEG, PNG, for example) and many printers. Thus a computer graphics programming library today need only support output to a workstation terminal or to a PostScript file.

The CALPLOT graphics package described here is robust, extensible and easy to use.

CHAPTER 1
CALPLOT GRAPHICS

1. Introduction

This chapter will introduce the concepts of CALPLOT programming and use. They are actually very simple. Programming computer graphics requires the same process as drawing a figure with a pen or pencil. There are just several stages: start the plot, select the pen, move the pen to the starting point on the drawing surface, lower the pen to draw, and finally declare that the effort is done. We first need to define the drawing surface.

As we will see later, the CALPLOT drawing surface can be very large, but the simplest approach is to envision a 10x8 grid on the computer screen as illustrated in the first figure.

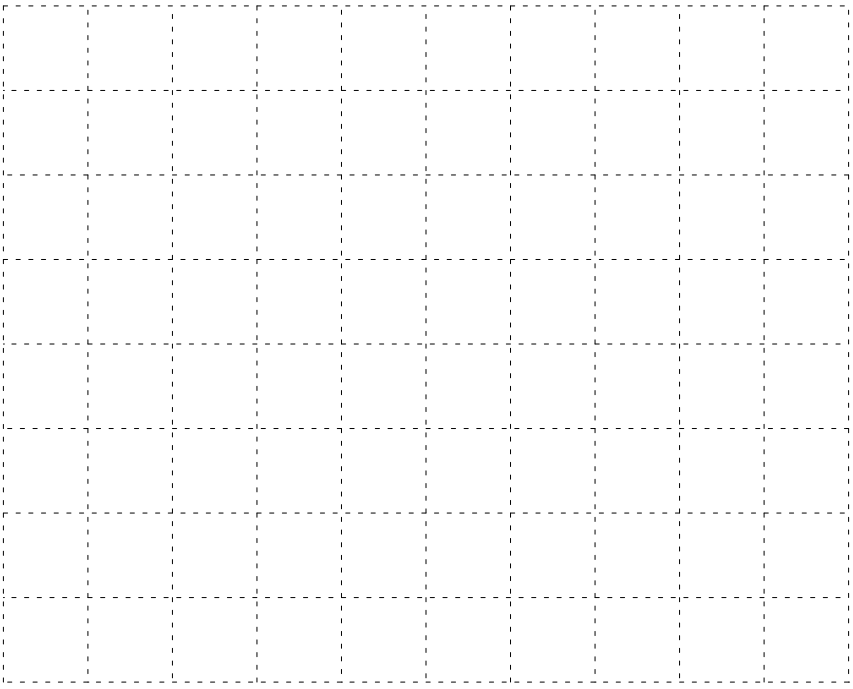


Fig. 1.1. The drawing grid. The lower left corner has the coordinates $(0, 0)$ and the upper right, $(10, 8)$.

Let's draw a rectangle with lower-left corner at $(2, 2)$ and upper-right corner at $(4, 3)$. The C and FORTRAN programs to do this are displayed in Table 1.1.

Table 1.1. Plot a rectangle

C

```
#include "calplot.h"
/*
 * cprogram to plot a rectangle
*/

void dorect(float xl, float yl,
   float xh, float yh);

main()
{
/*
 * initialize graphics
 */
   pinitf("RECT.PLT");
/*
 * make the line width 0.1 inches = 0.254cm
 */
   gwidth(0.1);
   dorect(2.0,2.0,4.0,3.0);
/*
 * terminate plot
 */
   pend();
}

void dorect(float xl, float yl,
   float xh, float yh)

{
/*
 * draw a rect move to (xl,yl) and
 * continue around the periphery
 */
   gmove(xl,yl);
   gcont(xh,yl);
   gcont(xh,yh);
   gcont(xl,yh);
   gcont(xl,yl);
}
```

FORTRAN

```
c-----
c  program to plot a rectangle
c-----
c-----
c  initialize graphics
c-----
   call pinitf('RECT.PLT')
c-----
c  make the line width 0.1 inches = 0.254cm
c-----
   call gwidth(0.1)
   call dorect(2.0,2.0,4.0,3.0)
c-----
c  terminate plot
c-----
   call pend()
   end

   subroutine dorect(xl,yl,xh,yh)
   real xl, tl, xh, yh
c-----
c  draw a rect  move to (xl,yl) and
c  continue the periphery
c-----
   call gmove(xl,yl)
   call gcont(xh,yl)
   call gcont(xh,yh)
   call gcont(xl,yh)
   call gcont(xl,yl)
   return
   end
```

As you can see, the code is heavily documented with comments, the programming is modular for generality and understanding, and the C and FORTRAN codes are very similar.

The first operation is to initialize graphics with the *pinitf()* call which also states that the graphics output will be in the file named *RECT.PLT*. We next set the line width of the following plot calls to be *0.1 inch* (which is equal to .254 cm). After this, we draw the rectangle by moving to the lower left corner, lowering the pen and continuing counter-clockwise back to the starting point. Finally the graphic output is terminated with the *pend()* command. Do not worry yet about inches or centimeters, just recall that the display window on the computer is always 10 x 8 and that we requested a line width of 0.1, or 1/100'th of the screen window width. If we actually plot this onto a piece of paper then the output will have the proper physical dimensions.

The next step is to compile and execute the program. I assume that the *CALPLOT* FORTRAN and C libraries have already been created and are available in the current directory. Normally these will be installed in one location on the system, and will be

accessed by defining their absolute or relative path. Since the working environments will be UNIX/LINUX/MacOSX or Windows we would do the following assuming that we use the gcc/g77 GNU compliers:

Table 1.2. Compile and Execute Program

C	FORTRAN
UNIX/LINUX/MacOSX:	
gcc rect.c libcalpltc.a -lm -o rect rect	g77 rect.f libcalpltf.a -o rect rect
Windows:	
gcc rect.c libcalpltc.a -lm -o rect.exe rect.exe	g77 rect.f libcalpltf.a -o rect.exe rect.exe

As a result of a successful compile and execution you will see the file *RECT.PLT* in the current directory if you use the UNIX *ls -l* or the WINDOWS Command tool command *dir*. The file *RECT.PLT* is a binary file of 59 bytes, which makes it very compact, containing commands for pen movement. The contents can be seen using the UNIX command *od -c RECT.PLT*:

```
[rbh Chap1]$ od -c < RECT.PLT
0000000   s  \0  \0  \0  \0 304 035 304 035   k  \0  \0 377 377 377 377
0000020 377 377 377 377   W  \0  \0   W  \0  \0   m  \0  \0  \0  \0   W
0000040   d  \0   m 320  \a 320  \a   n 240 017 320  \a   n 240 017 270
0000060  \v   n 320  \a 270  \v   n 320  \a 320  \a
0000073
[rbh Chap1]$
```

which is not very informative. Alternatively we can use the *CALPLOT* command *plotdbg* which reads the binary file and gives more meaningful information:

```
[rbh Chap1]$ plotdbg < RECT.PLT
Open plot.
Space  (0,0) to (7620,7620)
Reset Clip
Width: 0
Width: 0
Move to (0,0).
Width: 100
Move to (2000,2000).
Continue line to (4000,2000).
Continue line to (4000,3000).
Continue line to (2000,3000).
Continue line to (2000,2000).
Close plot.
[rbh Chap1]$
```

Note that the binary coordinates are such that 0.001 inch = 1 *CALPLOT* internal coordinate. This precision is sufficient for most graphics. The binary file initializes the graphics,

sets the linewidth and then draws the rectangle.

To view the results, we must use a program that converts the binary commands to output device specific code. Some choices are

```
plotxvig < RECT.PLT   (for X11 graphics on UNIX/LINUX/MacOSX)
plotmsw  < RECT.PLT   (for WIN32 on a WINDOWS PC)
plotnps  < RECT.PLT > rect.ps  (to create PostSCript)
plotnps -EPS  < RECT.PLT > rect.eps  (to create Encapsulated PostSCript)
```

There is also support for a few other graphics output devices: Tektronix (**`plot4014`**), PNG (**`plotpng`** on LINUX) and GIF (**`plotgif`**).

The following figure shows the rectangle created by the program, superimposed an the 8 x 10 reference grid.

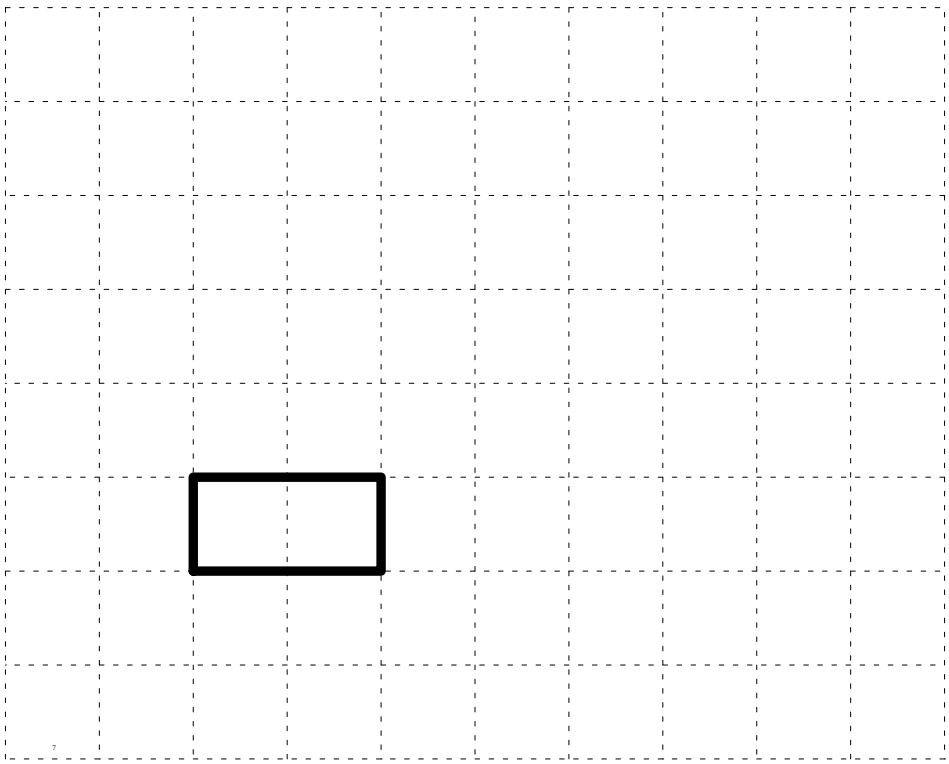


Fig. 1.2. The RECT.PLT superimposed on the drawing grid.

The line drawing primitive is the basis for drawing characters or numbers, since these can be composed of individual pen strokes. For better appearance special routines can be developed to use hardware fonts. Another primitive that is best implemented in hardware is polygon shading. To illustrate the output of text and polygon shading, the next example fills the rectangle with the color red and also places a caption at the top of the figure. The C code is shown in Table 1.3; the FORTRAN code would be similar in appearance. Execution of the code creates the new plot file *MRECT.PLT* which is superimposed on the drawing grid in Figure 1.3.

Table 1.3. Modified program to shade the rectangle

```
#include "calplot.h"
/*
 * c program to plot a rectangle
*/

void dorect(float xl, float yl, float xh, float yh);
void dopolyfill(float xl, float yl,
     float xh, float yh);
#define BLACK 1
#define RED 2

main()
{
/*
 * initialize graphics
 */
     pinitf("MRECT.PLT");
/*
 * fill a rectangle with the color RED
 */
     newpen(RED);
     dopolyfill(2.0,2.0,4.0,3.0);
/*
 * make the line width 0.1 inches = 0.254cm
 */

     gwidth(0.1);
     newpen(BLACK);
     dorect(2.0,2.0,4.0,3.0);
/*
 * label the rectangle - center on the rectangle
 * and note that CALPLOT letter width = height
 * Also return to default line width
 */
     gwidth(0.0);
     symbol(3.0 -4*0.10,3.2,0.10,"Red Fill",0.0,8);
/*
 * terminate plot
 */
     pend();
}

void dorect(float xl, float yl, float xh, float yh)
{
/*
 * draw a rect my move to (xl,yl) and continue
 *  around the periphery of the rect
 */
     gmove(xl,yl);
     gcont(xh,yl);
     gcont(xh,yh);
     gcont(xl,yh);
     gcont(xl,yl);
}

void dopolyfill(float xl, float yl,
     float xh, float yh)
{
     float xp[4], yp[4];
     int np;
/*
 * draw a rect my move to (xl,yl) and continue
 *  around the periphery of the rect
 */
     np = 4;
     xp[0] = xl; yp[0] = yl;
     xp[1] = xh; yp[1] = yl;
     xp[2] = xh; yp[2] = yh;
     xp[3] = xl; yp[3] = yh;
     shadep(np,xp,yp);
}
```

The sequence of drawing is important. Note that we first draw the shaded area, and then place the black edge around it. We also center the caption "Red Fill" by using the fact that the *symbol* is designed to have an inter-character space equal to the height. To center the caption, the desired center position was selected as *(3.0, 3.2)*, the number of characters in "Red Fill" was counted, and the starting position of the string was moved to the position *(2.6, 3.2)*. This logic employed is identical to that used if you were to center a phrase manually. The *symbol* routine always starts at the *(x, y)* coordinate given. We will see later in Chapter 5, that we can create a procedure called *gcent()* that will accomplish this task without manual computation of the start position.

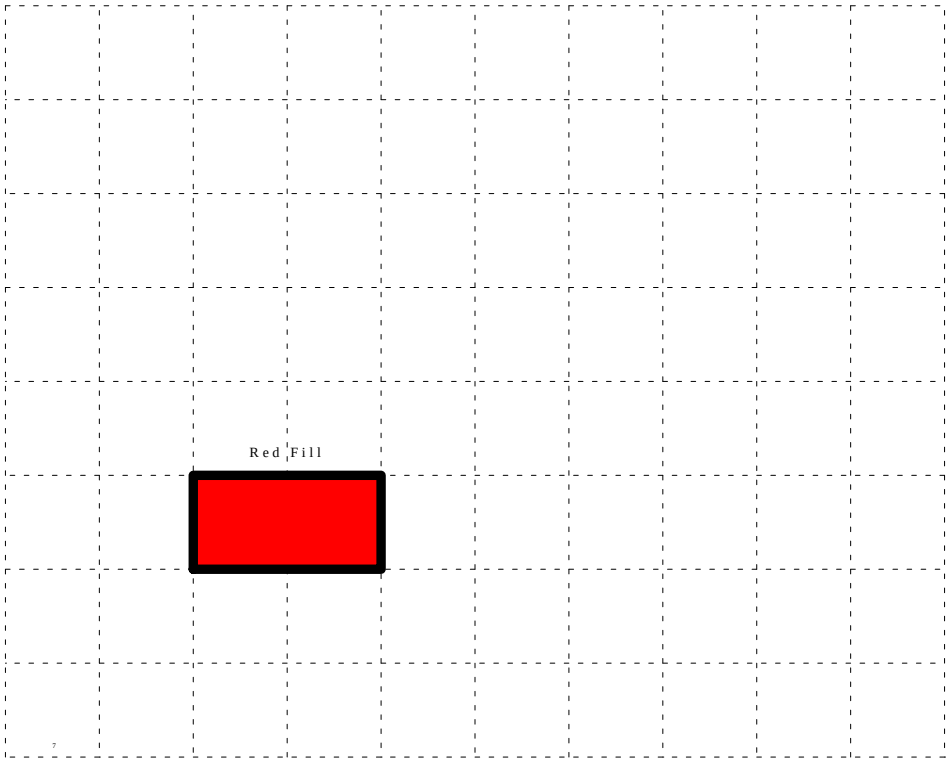


Fig. 1.3. The MRECT.PLT superimposed on the drawing grid.

The primitives *pinitf*, *plot*, *pend*, *symbol* and *shade* are all that are required to write your own computer graphics code.

CHAPTER 2
FORTRAN LIBRARY

This chapter introduces the FORTRAN language interface to the *CALPLOT* libraries. The chapter first provides a simple overview of the routines called in FORTRAN. Next a detailed description of the syntax for each user routine is provided. Finally, a set of sample programs, together with graphic output, is provided to demonstrate syntax as well to test the functioning of the library.

1. Introduction

In the description of the library calls, several different coordinates will be used. The term user values describes the actual numbers that the user works with. For example in plotting temperature versus time, the user may work with °*C* and *sec*. The use of the routines *pltscl(), algaxe(), pltlgd(), pltlog()* enable the user to not worry directly about the mapping of the user values to display coordinates. In particular, the use of the interactive routine *curuxy* in conjunction with these four routines, relieves the user of having to transform interactive screen coordinates back into user values.

The next term is user coordinates. This indicates the physical plotting coordinate on the device, which may be in units of cm or in according to the argument of the gunit() call. For interactive analysis, the user may get absolute display coordinates from curaxy(), or coordinates relative to the current origin and current plot scaling factor from the call currxy().

2. Library Calls

This section presents the *CALPLOT* library calls. Each section gives the purpose, the syntax and other helpful notes. In addition, reference is made to sample programs which better illustrate the use of the command.

The library calls supported and the general syntax are as follow:

```
call algaxe(xaxlen,yaxlen,nocx,nocy,ttlx,ttly,mtx,mty,x1,y1,deltax,deltay)
call axis(xpage,ypage,title,nchar,axlen,angle,firstv,deltav)
call curaxy(xx,yy,ic)
call curixy(ix,iy,ic)
call currxy(xx,yy,ic)
```

```
call curuxy(xx,yy,x1,y1,deltax,deltay,nocx,nocy,ic)
call factor (fact)
call frame()
call gcont(xx,yy)
call gclip(cmdstr,xlow,ylow,xhish.yhigh)
call gcursor(curstrstr)
call gend(mode)
call gframe(mode)
call gfont(ifont)
call ginfo(HasMouse, XminDev, YminDev, XmaxDev, YmaxDev,
     XminClip, YminClip, Xmaxclip, YmaxClip, Color)
call ginitf(plotstr, iconstr)
call gmesg(mesgstr)
call gmove(xx,yy)
call gread(fnamestr, X0, Y0, Xlow, Ylow, Xhigh, Yhigh,
     PageNum, scalx, Scaly)
call grdtxt(text,lstr)
call gunit(str)
call gwidth(width)
call gwrtxt(xx,yy,text,flgbln)
call line(x,y,n,inc,lintyp,inteq)
call lined(x,y,n,inc,lintyp,inteq,ipat,xlen)
call newpen(ipen)
call number(xpage,ypage,height,fpn,angle,ndec)
call pinit()
call pinitf(string)
call plot(x,y,ipen)
call plotd(xx,yy,ipat,xlen)
call plots(ibuf,nbuf,ldev)
call pltlgd(x,y,n,x1,y1,deltax,deltay,lintyp,inteq,ht,nocx,nocy,ipat,xlen)
call pltlog(x,y,n,x1,y1,deltax,deltay,lintyp,inteq,ht,nocx,nocy)
call pltscl(x,axlen,n,x1,deltax,nocx)
call gscale(x,xaxlen,n,inc)
call shadep(narr, xarr, yarr)
call shader(x1,y1,x2,y2,ipatx,ipaty,xlen,ylen)
call shadet(x1,y1,x2,y2,x3,y3,ipatx,ipaty,xlen,ylen)
call shdsei(x1,y1,ixy,istnd,iplmn)
call symbol (xpage,ypage,height,inbuf,angle,nchar)
call where(xpage, ypage, fct)
```

The following sections provide the detailed description of each routine.

algaxe

Use *algaxe* to draw logarithmic or linear axes at the current origin.

```
call algaxe(xaxlen,yaxlen,nocx,nocy,ttlx,ttly,mtx,mty,x1,y1,deltax,deltay)
```

real*4 xaxlen *length of x-axis in user units (inch or cm). If <=0 do not plot*
real*4 yaxlen *length of y-axis in user units (inch or cm). If <=0 do not plot*
real*4 x1 *First value in x-axis*
real*4 y1 *First value in y-axis*
real*4 deltax *Number of x-units per user unit (inch or cm) if linear*
Length of cycle in user units (inch or cm) if logarithmic
real*4 deltay *Number of y-units per user unit (inch or cm) if linear*
Length of cycle in user units (inch or cm) if logarithmic
character ttlx*(*) *x-axis title*
character ttly*(*) *x-axis title*
integer*4 mtx *Number of characters in x-axis title*
integer*4 mty *Number of characters in y-axis title*
integer*4 nocx *Number of logarithmic cycles on x-axis. If == 0, linear plot*
integer*4 nocy *Number of logarithmic cycles on y-axis. If == 0, linear plot*

See sample program *tstcur.f*

axis

Draw a linear axis scale. This is an original CALCOMP call.

```
call axis(xpage,ypage,title,nchar,axlen,angle,firstv,deltav)
```

real*4 xpage *x-coordinate of starting point of axis in user units*
(inch or cm) relative to current origin
real*4 ypage *y-coordinate of starting point of axis in user units*
(inch or cm) relative to current origin
real*4 axlen *Length of axis in user units (inch or cm)*
real*4 angle *Angle (degrees) that axis makes with the x-axis*
real*4 firstv *Value of axis at (xpage, ypage)*
real*4 deltav *Change in axis value between tic marks per unit*
length (inch or cm). If deltav < 0, the value at (xpage, ypage)
is a maximum, and values decrease along the axis
character title*(*) *Axis title*
integer*4 nchar *Number of characters in title. >0 for tic marks,*
numbers and title on counterclockwise side. <0 for tic marks,
numbers and title on clockwise side.

See the program example *line.f*.

curaxy

For terminals with cursor input, turn the cursor on, and acquire the coordinates of the cursor when any character is typed on the terminal. Return this character to the calling program.

```
call curaxy(xx,yy,ic)
```

real*4 *xx *x-coordinate of cursor in user coordinates*
real*4 *yy *y-coordinate of cursor in user coordinates*
character ic*2 *character returned with cursor coordinates. Must be defined as character ic*2*

If this is used with a library that does not support interactive cursor input, then the values returned are *xx = 0.0, yy = 0.0*, and *ic = ' '*. See sample program *tstcrs.f*

curixy

For terminals with cursor input, turn the cursor on, and acquire the coordinates of the cursor when any character is typed on the terminal. Return this character to the calling program. The coordinates returned are in the pseudo-screen coordinates relative to an origin at the lower left corner. The coordinates are effectively 0.001 inches.

```
call curixy(ix,iy,ic)
```

integer*4 *ix*x-coordinate of cursor in plot coordinates*
integer*4 *iy *y-coordinate of cursor in plot coordinates*
character ic*(*) *character returned with cursor coordinates. Must be defined as character ic*2*

If this is used with a library that does not support interactive cursor input, then the values returned are *ix = 0, iy = 0*, and *ic = ' '*. This is not too useful for the user, but this primitive is required to support the other three cursor input routines. See sample program *tstcrs.f*

currxy

For terminals with cursor input, turn the cursor on, and acquire the coordinates of the cursor when any character is typed on the terminal. Return this character to the calling program. The coordinates returned are user coordinates relative to the current origin.

```
call currxy(xx,yy,ic)
```

real*4 *xx *x-coordinate of cursor in user coordinates*
real*4 *yy *y-coordinate of cursor in user coordinates*
character ic*(*) *character returned with cursor coordinates. Must be defined*

*as character ic*2*

If this is used with a library that does not support interactive cursor input, then the values returned are *xx = 0.0, yy = 0.0*, and *ic = ' '*. See sample program tstcrs.f

---

curuxy

This procedure is used for interactive cursor graphics. By defining the parameters and drawing with algaxe, pltlog, pltscl and pltlgd, the screen coordinates are converted to user coordinates.

```
call curuxy(xx,yy,x1,y1,deltax,deltay,nocx,nocy,ic)
```

real*4 *xx	*x-coordinate of cursor in user coordinates*
real*4 *yy	*y-coordinate of cursor in user coordinates*
real*4 x1	*User value of left side of x-axis*
real*4 y1	*User value of bottom side of y-axis*
real*4 deltax	*Number of x-units per user unit (inch or cm) if linear* *Length of cycle in user units (inch or cm) if logarithmic*
real*4 deltay	*Number of y-units per user unit (inch or cm) if linear* *Length of cycle in user units (inch or cm) if logarithmic*
integer*4 nocx	*Number of logarithmic cycles on x-axis. If == 0, linear plot*
integer*4 nocy	*Number of logarithmic cycles on y-axis. If == 0, linear plot*
character ic*(*)	*character returned with cursor coordinates. Must be defined as character ic*2*

If this is used with a library that does not support interactive cursor input, then the values returned are *xx = 0.0, yy = 0.0*, and *ic = ' '*. See sample program *tstcur.f*

---

factor

Use *factor* to scale all plot moves by *fact*. *fact < 1.0* **makes the plot uniformly smaller, while** *plot > 1.0* makes the plot uniformly larger than normal. This is an original CAL-COMP call.

```
call factor (fact)
```

real*4 fact *Multiply all moves by* fact

frame

Start plot on a new page. On hardcopy output, the paper is advanced. On a graphics terminal, the display is erased.

```
call frame()
```

See sample program *tabl.f.*

gclip

Define a clip region for plots.

```
gclip(cmdstr, xlow, ylow, xhigh, yhigh)
character cmdstr*(*)
real xlow, xhigh, ylow, yhigh
```

Define a clipping rectangle bounded by the coordinates (xlow,ylow) (xhigh, yhigh) if cmdstr = 'ON' or remove all clip regions is cmdstr = 'OFF' See sample programs *button.f*, *new.f* or *stars.f.*

gcont

Draw a line in the current color from the current point to the coordinates (xx,yy) Start plot on a new page. On hardcopy output, the paper is advanced. On a graphics terminal, the display is erased.

```
gcont(xx,yy)
real xx, yy
```

gcursor

Define the cursor to be used for interactive programs. The cursor can either be a hardware cursor, or an XOR cursor. The hardware cursor will always be seen, but the XOR color may be difficult to see with some colors.

Although hardware cursor if preferred since it can be easily seen against all backgrounds, interesting effects can be created if a different cursor is combined with the rectangular region defined by the **`gclip()`** call. In this case the hardware cursor appears outside the rectangular clip region and the software cursor within.

```
gcursor(cursorstr)
```

```
character cursorstr*(*)
```

The character string defines the cursor type:

'Arrow'	Hardware arrow cursor
'Xorarrow'	XOR arrow cursor
'Crosshair'	XOR crosshair
'Plus'	XOR plus
'Box'	Draw a box (see cdraw.c example)
'Rubber'	Draw rubber band line (see cdraw.c example)
'Off'	Return to previous cursor (this only works back to one level)

The sample program *new.f* exercises the cursor. It also demonstrates how the cursor works when a rectangular clip region is defined. For example, one can have a crosshairs within a rectangular region but the cursor reverts to the hardware arrow cursor outside this region.

---

gend

Gracefully terminate the plotting session. this is an extension of *pend()* which is now *gend(0)*. For interactive graphics drivers, a *gend(1)* forces a termination of the graphics window. The use will not have to manually hit the **`Quit`** button.

```
call gend(mode)
integer mode
```

This is meant to replace the old *plot(0.0,0.0,999)*.

---

gfont

Change the font in use by the *symbol()* call. By definition there is no upper limit to the number of fonts used. All plot drivers support the following. See the documentation for the specific plotting device for more detail.

Fonts Supported

Font Number	Font
0	Hardware default or override
1	Regular
2	Italic
3	Bold

4 Greek

```
call gfont(ifont)
integer*4 ifont Font value
```

For generality, a font value of 9 would reduce to a value of 1 on a device that supports only the four fonts. Thus, fonts 2, 6, 10, etc., always refer to an italic font, and 3, 7, 11, etc., always will give a bold font.

The sample program *tabl.f* makes use of the *gfont(0)* call to permit a quick perusal of all fonts at plot time. For example if the mapping of symbol number is desired in Helvetica-Bold, one plots the plot file generated by this program, *TABL* on the PostScript printer by the command

plotnps -F7 < TABL > PostScript_Language_File

See the test program *gwid.f*.

gframe

This is an extension of the frame() command. Normally a call for a page erase is only performed after the user manually clicks the mouse or hits the *Next* button at the top of the *plotmsw* or *plotxvig* presentation. The *gframe* permits an automatic move to the next page. The programmer can construct a button which is used to proceed to the next page.

```
call gframe(mode)
integer mode
```

If *mode = 1*, automatically go to the next page, else wait for user interaction.
If *mode = 2*, if the screen was resized, redraw the window to the new CALPLOT 10x8 size, and then automatically go to the next page.

ginfo

Obtain information about the graphics state. this information can be use to develop alternative presentations according to available screen area. For example, one may use hardware fonts if there is enough space, but revert to simple vector fonts for very small work areas.

```
ginfo(HasMouse, XminDev, YminDev, XmaxDev, YmaxDev,
     XminClip, YminClip, Xmaxclip, YmaxClip, Color)
integer HasMouse
Real XminDev, YminDev, XmaxDev, YmaxDev
```

```
real XminClip, YminClip, Xmaxclip, YmaxClip
integer Color
```

HasMouse	1 for true, 0 for false
XminDev, YminDev	Display size in pixels
XmaxDev, YmaxDev	
XminCLip, YminClip	Current clip region in user
XmaxClip, YmaxClip	units
Color	

See sample program *tabl.f.*

---

ginitf

Generalization of *pinitf*. Initialize plot. For non-interactive displays, a binary graphics meta-file which is used as input to the **calplot** commands. On an graphics terminal, this will initiate the graphics display. Up to the invocation of this call, the user may use C language IO to the screen. But once graphics are initialized, the user must use the *gwrtxt()* and *grdtxt()* to avoid any interference with the display.

When a plot meta-file is created, instead of creating the default plotXXXXX file as in *pinit()*, the user may control the naming and the redirection of output.

```
ginitf(plotstr,iconstr)
character plotstr*(*)      Name of plot file
    If plotstr = 'plot', open a unique file named plotXXXXX
    If plotstr = 'stdout', write the plot command stream to
    the standard output for redirection or piping into the
    calplot command
            If plotstr = 'INTER' enter interactive mode if supported by
                library
    If plotstr = 'anything else', the output will be placed in a
    file named 'anything else'.
character iconstr*(*)  name of title to be placed on the
            window manager frame, or when iconified.
```

An interesting aspect is that one can have an interactive plot, then redirect the graphics to an output file, and then return to the interactive plot. This is indicated in the test program *button.f.*

iconstr is the name that should appear on the taskbar of the window in an interactive program. This is useful if the window is minimized so that the program can be restarted from the taskbar See sample program *button.f.*

gmesg

Put an informative message on the top bar of the X11 or Windows display. This is very useful in an interactive program for presenting information or program status.

```
gmesg(mesgstr)
character mesgstr*(*)
```

See sample program *new.f* or *clip.f*.

gmove

Lift the pen and move to the coordinates (xx,yy)

```
gmove(xx,yy)
real xx, yy
```

grdtxt

Read a text string from the screen while in graphics mode. The input string is echoed, and a user defined erase key erases errors (For the PC this is a BackSpace). The input string is read until either a RETURN is entered, or until the input string length is exceeded.

For safety with any input, the string should be blank filled prior to input to remove the dregs of previous input.

```
call grdtxt(text,lstr)
character text*(*)
```
String to be returned to program

integer*4 lstr *Length of string*

See sample program *gphtxt.f*.

gread

Start plot on a new page. On hardcopy output, the paper is advanced. On a graphics terminal, the display is erased.

```
gread(fnamestr, X0, Y0, Xlow, Ylow, Xhigh, Yhigh,
     PageNum, scalx, Scaly)
character fnamestr*(*)
```

```
real X0, Y0
real Xlow, Ylow, Xhigh, Yhigh
integer PageNum
real scalx, scaly
```

See sample program *gread.f*.

---

gunit

All plot movements after this point will be in the units defined. Initially all movements are in inches.

```
call gunit(str)
character str*(*)Either the strings 'CM', 'cm', 'in' or 'IN'
            call gunit('in')/* draw a square */
            call plot(0.0,0.0,-3)
            call plot(1.0,0.0,2)
            call plot(1.0,1.0,2)
            call gunit('cm')
            call plot(0.0,2.54,2)
            call plot(0.0,0.0,2)
```

---

gwidth

Make the width of all plotted lines *width* user units.

```
call gwidth(width)
real*4 widthWidth of all lines drawn in user units
```

A value of *width = 0.0* will be interpreted as the minimally resolvable width, and not as a blank line. Since drawing a wide line takes time on most devices, the plot drivers have an option to turn off line width. See sample program *gwid.f*.

---

gwrtxt

Write graphic text to the screen, using hardware character capability if possible. The text is displayed horizontally parallel to the x-axis.

```
call gwrtxt(xx,yy,text,flgbln)
real*4 xx     (xx,yy) are coordinates of lower left corner of string
real*4 yy
character text*(*)Text to be displayed, NULL terminated
integer*4 flgbln  0 - write text out
```

1 - overwrite existing output with blanks prior to write,
filling with the background color of the display
2 - display output in reversed video mode

For almost all displays, no more than 80 characters can be displayed horizontally.

The text is written in using the current pen color defined by the last call to *newpen()* with argument < 1000. In the reversed video mode, the default newpen(1) color will define the background in the display box, and the normal background will be used for the text. In general, this will yield a white writing on a black background for laser or printer output. See test program *gphtxt*.

---

line

Draw a line, optionally using symbols to represents data points. This is an original CAL-COMP call.

`call line(x,y,n,inc,lintyp,inteq)`

`real*4 x(*)` *Array of abscissa*
real*4 y(*) *Array of ordinates*
integer*4 n *Number of points in array to be plotted*
integer*4 inc *plot every* inc'th *point*
there are n*inc + inc + 1 *points in the array*
integer*4 lintyp *!=0 plot symbol* inteq*at each abs*(lintyp) *point*
=0 plot line only
<0 plot only symbols, no connecting line
>0 plot symbols, with connecting line
integer*4 inteq *If* lintyp != 0, *plot this symbol*

The mapping of the value of *inteq* to a specific symbol is shown by the test program *tabl.f*.

The arrays *x(*)f1 and y(*)* must have the *firstv* and *deltav* entries placed into them by the *scale()* routine. Note also the need for additional array storage beyond *n*. See test program *testl.f*.

---

lined

`call lined(x,y,n,inc,lintyp,inteq,ipat,xlen)`

`real*4 x(*)` *Array of abscissa*
real*4 y(*) *Array of ordinates*
integer*4 n *Number of points in array to be plotted*
integer*4 inc *plot every* inc'th *point*

there are n*inc + inc + 1 *points in the array*

integer*4 lintyp *!=0 plot symbol* inteq*at each abs*(lintyp) *point*
=0 plot line only
<0 plot only symbols, no connecting line
>0 plot symbols, with connecting line

integer*4 inteq *If* lintyp != 0, *plot this symbol*

integer*4 ipat *Binary pattern for display (0,31)*

real*4 xlen *Length in user units of each bit of the pattern*

The mapping of the value of *inteq* to a specific symbol is shown by the test program *tabl.f*.

The arrays *x(*)f1 and y(*)* must have the *firstv* and *deltav* entries placed into them by the *scale()* routine. Note also the need for additional array storage beyond *n*. See test program *testld.f*.

---

newpen

Select pen color for the plot. This is an original *CALPLOT* call.

```
call newpen(ipen)
integer*4 ipen
```
Pen color

The mapping of colors is hardware dependent. *ipen=0* indicates the background color, which is not useful, unless one wishes to overwrite previous out, especially with shading. *0 < ipen < 1000* indicates a color. Hardware devices with just 4 pen colors, will use a mod operator to map, *ipen=9* into the color for *ipen=1*. The values *1000 <= ipen <= 1100* are used to define halftone or color fill patterns for the *shader()* and *shadet()*. See sample program *plttst.f* for color selection. See sample programs *gryscl.f, grytst.f,* or *nseitst.f* for colored (halftone) shading.

---

number

Convert a floating point number to a character string, and plot the string (number). This is an original *CALCOMP* call that has been extended to also use computer and scientific notation.

```
call number(xpage,ypage,height,fpn,angle,ndec)
real*4 xpage
```
x-coordinate of first lower left corner of number
xpage = 999.0 *continues with* xpage *set to the final x-coordinate from the last call to symbol or number*

real*4 ypage *y-coordinate of first lower left corner of number*
ypage = 999.0 *continues with* ypage *set to the final y-coordinate from the last call to symbol or number*

real*4 height	*Height of each character plotted*
real*4 fpn	*Number to be plotted*
real*4 angle	*Plot number at* angle *degrees with respect to x-axis*
integer*4 ndec	*Numerical precision of number*
	ndec - *number of figures to the right of the decimal place*
	If ndec = -1, *the floating point number will appear as*
	an integer. No decimal place is plotted.
	If ndec >= 1000 and <= 1009, *exponential FORTRAN format is invoked*
	of the form Ew.d. *The precision* d *is set by defining*
	ndec = 1000 + d. *Note that all numbers are left justified.*
	For this exponential format, the total width is (d + 6) * height**.**
	If ndec >= 2000 and <= 2009, *Scientific notation is used*
	The mantissa has a precision given by d, *set by defining*
	ndec = 2000 + d. *Note that all numbers are left justified.*
	For this exponential format, the total width about (d + 6)*height**.**

The use of *ndec >=1000* is an extension of the original definition made for these libraries.

The convention for character size, is that the *height* be the distance from the baseline to the maximum symbol elevation, for example the distance from the base to the top if the letters **I** or **A**. The convention for the inter character spacing is that the character with be equal to the height, with the actual character filling *0.6 * height* of this space. Thus the lower left corners of the two **2**'s in **22** are separated by *height* units, and the space between the two characters is the *0.4 * height*.

See sample program *tabl.f*.

---

pend

Gracefully terminate the plotting session.

```
call pend()
```

This is meant to replace the old *plot(0.0,0.0,999)*.

---

pinit

Initialize plot. For non-interactive displays, a **plotXXXXX** file which is used as input to the **calplot** commands. On an graphics terminal, this will initiate the graphics display. Up to the invocation of this call, the user may use C language IO to the screen. But once graphics are initialized, the user must use the *gwrtxt()* and *grdtxt()* to avoid any

interference with the display.

```
call pinit()
```

---

pinitf

Open the Initialize plot. For non-interactive displays, a binary graphics meta-file which is used as input to the **calplot** commands. On an graphics terminal, this will initiate the graphics display. Up to the invocation of this call, the user may use C language IO to the screen. But once graphics are initialized, the user must use the *gwrtxt()* and *grdtxt()* to avoid any interference with the display.

When a plot meta-file is created, instead of creating the default plotXXXXX file as in *pinit()*, the user may control the naming and the redirection of output.

```
call pinitf(string)
character string*(*)
```
Name of plot file
If string = 'plot', open a unique file named plotXXXXX
If string = 'stdout', write the plot command stream to the standard output for redirection or piping into the calplot *command*
If string = 'anything else', the output will be placed in a file named 'anything else'.

All test programs direct output to a file for that program, e.g., *tabl.f* creates a plot file *TABL*.

---

plot

Move the pen from the current position to the given coordinates. This is an original CALCOMP call.

```
call plot(x,y,ipen)
real*4 x
```
(x,y) *are the desired position*

real*4 y

integer*4 ipen *-3 - pen up during move (hidden vector), establish new origin at end of movement*
-2 - pen down during move (drawn vector), establish new origin at end of movement
+2 - pen down during move (drawn vector), do not establish new origin at end of movement
+3 - pen up during move (drawn vector), do not establish new origin at end of movement
999 - terminate plot after the move. For compatibility only, use pend()

plotd

This connects the current point and the point *(xpage, ypage)* with a dashed line given by the pattern *ipat*. The attached example tests this routine. The idea is that the integer *ipat* is a number between 0 and 31 which defines a five bit pattern, each of which represents a pen movement of *xlen* units. As each segment is plotted, the next bit in *ipat* is interrogated whether it is 0 or 1. A 1 represents a plotted vector, and a zero, an unplotted vector. Because a sixth bit is implicitly assumed to be zero, certain patterns are just a linear shift of others, e.g., 1 and 2, 1 and 4, etc. The unique patterns available have *ipat* values of *0, 1, 3, 5, 7, 9, 11, 13, 15, 21, 23, 27, and 31*. The attached figure shows this correspondence.

```
call plotd(xx,yy,ipat,xlen)
```

real*4 xx *x-coordinate at end of line*
real*4 yy *y-coordinate at end of line*
real*4 xlen *Length in user units of each bit of the pattern*
integer*4 ipat *Binary pattern for display (0,31)*

See sample program *dtest.f*.

plots

Initialize plot stream. This is an original CALCOMP call. However, the use of *pinitf* or *ginitf* is preferred.

```
call plots(ibuf,nbuf,ldev)
integer*4 ibuf
integer*4 nbuf
integer*4 ldev
```

This is used for upward compatibility of old programs. This routine just invokes *pinit()*.

pltlgd

A generalization of the *line()* routine, except that logarithmic scaling is supported. In addition, dashed lines are permitted. It differs from the *line()* routine in that each point is plotted, and that the coordinate arrays need not have scaling numbers put into them.

```
call pltlgd(x,y,n,x1,y1,deltax,deltay,lintyp,inteq,ht,nocx,nocy,ipat,xlen)
```

real*4 x(*) *Array of abscissa to be plotted*
real*4 y(*) *Array of ordinates to be plotted*
integer*4 n; *Number of points to be plotted*
real*4 x1 *If linear axis, this is the value at the left end*

of the x-axis. If logarithmic, this is the log10 *of the*
left end of the x-axis. This will be the exponent displayed.

real*4 y1 *If linear axis, this is the value at the left end*
of the y-axis. If logarithmic, this is the log10 *of the*
left end of the y-axis. This will be the exponent displayed.

real*4 deltax *If linear, this is the number of units per basic*
unit (inch or cm) along the x-axis. If logarithmic, this is the number
of basic units(inch or cm) per cycle of the logarithmic axis.

real*4 deltay *If linear, this is the number of units per basic*
unit (inch or cm) along the y-axis. If logarithmic, this is the number
of basic units(inch or cm) per cycle of the logarithmic axis.

integer*4 lintyp *!=0 plot symbol* inteq*at each abs*(lintyp) *point*
=0 plot line only
<0 plot only symbols, no connecting line
>0 plot symbols, with connecting line

integer*4 inteq *If* lintyp != 0, *plot this symbol*

real*4 ht *Height of symbol plotted on line*

integer*4 nocx *0 - x-axis is linear*
>0 - x-axis is logarithmic with this many cycles

integer*4 nocy *0 - y-axis is linear*
>0 - y-axis is logarithmic with this many cycles

integer*4 ipat *Binary pattern for display (0,31)*

real*4 xlen *Length in user units of each bit of the pattern*

See the sample program *tstcur.f*.

---

pltlog

A generalization of the *line()* routine, except that logarithmic scaling is supported. It differs from the *line()* routine in that each point is plotted, and that the coordinate arrays need not have scaling numbers put into them.

```
call pltlog(x,y,n,x1,y1,deltax,deltay,lintyp,inteq,ht,nocx,nocy)
```

real*4 x(*) *Array of abscissa to be plotted*

real*4 y(*) *Array of ordinates to be plotted*

integer*4 n; *Number of points to be plotted*

real*4 x1 *If linear axis, this is the value at the left end*
of the x-axis. If logarithmic, this is the log10 *of the*
left end of the x-axis. This will be the exponent displayed.

real*4 y1 *If linear axis, this is the value at the left end*
of the y-axis. If logarithmic, this is the log10 *of the*
left end of the y-axis. This will be the exponent displayed.

real*4 deltax *If linear, this is the number of units per basic*
unit (inch or cm) along the x-axis. If logarithmic, this is the number
of basic units(inch or cm) per cycle of the logarithmic axis.

real*4 deltay *If linear, this is the number of units per basic*

unit (inch or cm) along the y-axis. If logarithmic, this is the number of basic units(inch or cm) per cycle of the logarithmic axis.

integer*4 lintyp *!=0 plot symbol* inteq*at each abs*(lintyp) *point*
=0 plot line only
<0 plot only symbols, no connecting line
>0 plot symbols, with connecting line

integer*4 inteq *If* lintyp != 0, *plot this symbol*

real*4 ht *Height of symbol plotted on line*

integer*4 nocx *0 - x-axis is linear*
>0 - x-axis is logarithmic with this many cycles

integer*4 nocy *0 - y-axis is linear*
>0 - y-axis is logarithmic with this many cycles

See sample program *tstcur.f*.

---

pltscl

A generalization of the *call gscale* call to include establishing scaling for logarithmic plots. If *nocx* is zero, linear scaling is performed by a call to the subroutine *scale*, while if *nocx* is greater than 0, logarithmic scaling is performed to make a integral number of cycles. The plot is scaled to include the largest data value, lower values may not be plotted.

```
call pltscl(x,axlen,n,x1,deltax,nocx)
```

real*4 x(*) *Array to be scaled to fit axis length*

real*4 axlen *Length of axis*

integer*4 n *Number of points in the array*

real*4 *x1 *A value returned by the program. For a linear plot, this gives the user value at the lowest value of th axis, while for logarithmic plots this is the* log10 *of the lowest value.*

real*4 *deltax *Conversion from user values to user coordinates. For a linear plot, this is the number of user values per basic user coordinate (cm or inch). For a logarithmic plot, this is the number of basic user units (cm or inch) per cycle.*

integer*4 nocx *> 0 - number of logarithmic cycles along the axis*
0 - axis is linear

{ *> 0 - number of logarithmic cycles along the axis*
0 - axis is linear

See sample program *tstcur.f*.

gscale

This is used for automatic scaling of plot axes and for line call. array(i) is examined to choose a nice first value, firstv above, which is returned as array(npts) and a reasonable increment, deltav, which is returned as array(npts+abs(inc)). deltav is the number of array(i) units per inch. To obtain the plot space coordinates, programs compute actual movement = (array(i) - firstv)/deltav.

This is used in conjunction with *line()*.

The original CALPCOMP call *scale* is replaced by gscale since FORTRAN 90 preempts the use of the name *scale*.

```
call gscale(x,xaxlen,n,inc)
```

real*4 x(*)	*Array to be scaled*
real*4 xaxlen	*Length of axis on which array is to fit*
integer*4 n	*Number of points in the array*
integer*4 inc	*Increment for plotting, e.g., plot each abs(*inc*)* *point. A negative value indicates that the computed* firstv *should be a maximum rather than a minimum.*

See sample program *line.f*.

shadep

Shade a polygon with the current pen color

```
shadep(narr, xarr, yarr)
integer narr
real xarr(*), yarr(*)
```

See sample program *new.f*, *stars.f* and *grid.f*.

shader

This shades a rectangular area using different patterns in the x- and y-directions. The meaning of the pattern is described under *plotd*. The only real difference is that this pattern is generated in the device filter and not in the source level for efficiency. Thus this differs from *plotd* and *lined* in that the pattern is repeated with respect to the absolute page coordinate, hence the pattern within a rectangular region is the same, but shifted, when the rectangle is placed at different positions on the page. You may imagine this as having a uniform background pattern, and moving a window across.

`call shader(x1,y1,x2,y2,ipatx,ipaty,xlen,ylen)`

`real*4 x1` (x1,y1) *are one corner of the rectangle*
real*4 y1
real*4 x2 (x2,y2) *are opposite corner of the rectangle*
real*4 y2
integer*4 ipatx *bit pattern along the x-axis*
integer*4 ipaty *bit pattern along the y-axis*
real*4 xlen *length of each bit of the* ipatx *pattern along the x-axis*
real*4 ylen *length of each bit of the* ipaty *pattern along the y-axis*

If *ipatx = 0* and *ipaty = 0*, then a solid fill is made using the the current shading value defined by *newpen()*. See the test programs *pattst.f*, *gryscl.f*, and *grytst.f*.

---

shadet

This shades a triangular area using different patters in the x- and y- directions. The pattern plotted is defined under the description of plotd(). The only minor problem is that the plot pattern is generated in the hardware specific plot filter, and not in the source level for efficiency. Thus this differs from plotd() and lined() in that the pattern is defined with respect to the absolute page coordinate, hence the pattern within the triangular region is shifted within the triangle, when the triangle is moved to different positions on the page. The underlying pattern is fixed. In addition, the agreement of this pattern on different devices will not be perfect, due to different hardware resolution.

`call shadet(x1,y1,x2,y2,x3,y3,ipatx,ipaty,xlen,ylen)`

`real*4 x1` (x1,y1) *are one corner of the triangle*
real*4 y1
real*4 x2 (x2,y2) *are another corner of the triangle*
real*4 y2
real*4 x3 (x2,y2) *are third corner of the triangle*
real*4 y3
integer*4 ipatx *bit pattern along the x-axis*
integer*4 ipaty *bit pattern along the y-axis*
real*4 xlen *length of each bit of the* ipatx *pattern along the x-axis*
real*4 ylen *length of each bit of the* ipaty *pattern along the y-axis*

If *ipatx = 0* and *ipaty = 0*, then a solid fill is made using the the current shading value defined by *newpen()*. See the test program *tritst.f* for usage. In addition see the test programs *pattst.f* for more patterns and *dtest.f* for the bit patterns.

shdsei

This is a special routine designed to invoke shaded area plots of seismic traces, in a manner that is very efficient at the source level. The idea used is that this routine be invoked twice for each trace plotted, once to turn the shading on, and then after the trace is plotted, to turn the shading off. The center line of the trace must be defined in plot space coordinates. The hardware specific plot filter will use this information to shade the trace as it is plotted. The time axis of the trace can be parallel to either the x-axis or the y-axis. Positive or negative amplitude fill can be done.

```
call shdsei(x1,y1,ixy,istnd,iplmn)
```

real*4 x1 *x-coordinate of the first point of the trace, assuming that the first time sample has zero amplitude.*

real*4 y1 *y-coordinate of the first point of the trace, assuming that the first time sample has zero amplitude. If the trace is plotted parallel to the x-axis, then only the* y1 *value is meaningful since this states where the amplitude = 0 line is.*

integer*4 ixy *Flag to indicate whether the time axis is parallel to the x-axis* (0) *or the y-axis* (1)**.**

integer*4 istnd *Flag to indicate whether to turn strace shading on* (1) *or off* (0)**.**

integer*4 iplmn *(Flag to indicate whether positive amplitudes are to be shaded* (0) *or negative amplitudes* (1) *or both* (2)*. If trace is plotted parallel to x-axis, then positive amplitudes are y-values* > y1**.**

See the test programs *seitst.f* for simple usage and *nseitst.f* for use together with shading to display trace attributes in addition.

symbol

Use symbol to plot a character or special symbol at the current position. Two different syntaxes are used, according to whether a string is plotted or just a single character or special symbol.

This is an original CALCOMP call. However, the strict distinction between strings and numbers in FORTRAN 77, means that *inbuf* is a string and to select a special symbol defined by an integer *m*, one must use *inbuf = char(m)*.

```
call symbol (xpage,ypage,height,inbuf,angle,nchar)
```

real*4 xpage *x-coordinate of first lower left corner of number* xpage = 999.0 *continues with* xpage *set to the final x-coordinate from the last call to symbol or number*

real*4 ypage *y-coordinate of first lower left corner of number*
ypage = 999.0 *continues with* ypage *set to the*
final y-coordinate from the last call to symbol or number
real*4 height *Height of each character plotted*
real*4 angle *Plot string at* angle *degrees with respect to x-axis*
character inbuf*(*)*Character string to be plotted*
integer*4 nchar *If* nchar > 0 *plot* nchar *characters of the string*
If nchar < 0 *plot only a single symbol defined*
by the number stored in inbuf(0). *If* nchar = -1,
no line is drawn from the last position to this coordinate.
If nchar < -1, *then a connecting line is drawn.*
Finally, if nchar < 1, *the first 16 symbols will be*
centered at the coordinates (xpage, ypage).

The convention for character size, is that the *height* be the distance from the baseline to the maximum symbol elevation, for example the distance from the base to the top if the letters **I** or **A**. The convention for the inter character spacing is that the character with be equal to the height, with the actual character filling *0.6 * height* of this space. Thus the lower left corners of the two **A**'s in **AA** are separated by *height* units, and the space between the two characters is the *0.4 * height*. This information is given for positioning of characters. See the test program *tabl.f* for the mapping between *inbuf* and a specific symbol for negative values of *nchar*..

```
call symbol(1.0,1.0,0.20,'HELLO WORLD',0.0,11)
```
plots string parallel to x-axis starting at (1.0,1.0)

```
call symbol(2.0,2.0,0.10,'0',0.0,-1)
```
centers a square symbol at position (2.0,2.0)

---

where

Determine the current plotting point position in user coordinates. This is an original CALCOMP call.

```
call where(xpage, ypage, fct)
real*4 *xpage
```
x - coordinate of current position
real*4 *ypage *y - coordinate of current position*
real*4 *fct *A necessary parameter for upward compatibility*
Always returned as 1.0

3. Internal Routines

The following are some of the internal routines used by the *CALPLOT* libraries. The user need only note that these names are already taken, and that a user program must

avoid defining these names or, in general, using these names directly. Since the gcc/g77 compilers are used, the underlying C routines cannot be directly called from FORTRAN.

This section contains auxiliary routines and data structures required to implement the *CALPLOT* package. For the most part, users must not redefine any of these names or attempt to use them. Otherwise *CALPLOT* will not function as desired.

Common Names

These are common to all routines.

```
common/Gcplot/owidth
common/Scplot/x0,y0
common/Xcplot/xold,yold,xcur,ycur
common/Ycplot/xstp,ystp
common/Zcplot/iunit,ifont
```

Low level Subroutines

These are interfaces to the low level C routines.

```
call dfclip()
call dfclos()
call dfcont()
call dfcros()
call dfcurs()
call dferas()
call dffilp()
call dffilr()
call dffils()
call dffilt()
call dffont()
call dfgint()
call dfgott()
call dfgsym()
call dfgwid()
call dfinfo()
call dfmesg()
call dfmove()
call dfopen()
call dfpenn()
call dfread()
call dfspce()
```

CHAPTER 3
C LIBRARY

This chapter introduces the C language interface to the *CALPLOT* libraries. The chapter first provides a simple overview of the routines called in C. Next a detailed description of the syntax for each user routine is provided. Then a description is provided of data structures and internal routines that are necessary for the library operation, and whose names in general must not be used in a user program to avoid conflict with the libraries. Since some of the low level C routines are the same as the UNIX Version 7 *plot(3)* calls, they may still be used, even though the syntax may be slightly different. Finally, a set of sample programs, together with graphic output, is provided to demonstrate syntax as well to test the functioning of the library.

1. Introduction

In the description of the library calls, several different coordinates will be used. The term user values describes the actual numbers that the user works with. For example in plotting temperature versus time, the user may work with °*C* and *sec*. The use of the routines *pltscl(), algaxe(), pltlgd(), pltlog()* enable the user to not worry directly about the mapping of the user values to display coordinates. In particular, the use of the interactive routine *curuxy* in conjunction with these four routines, relieves the user of having to transform interactive screen coordinates back into user values.

The next term is user coordinates. This indicates the physical plotting coordinate on the device, which may be in units of cm or in according to the argument of the gunit() call. For interactive analysis, the user may get absolute display coordinates from curaxy(), or coordinates relative to the current origin and current plot scaling factor from the call currxy().

2. Library Calls

This section presents the *CALPLOT* library calls. Each section gives the purpose, the syntax and other helpful notes. In addition, reference is made to sample programs which better illustrate the use of the command.

The library calls supported and the general syntax are given in the function prototype include file *calplot.h*:

```
void algaxe(float xaxlen,float yaxlen,int nocx,int nocy,
```

```
   char *ttlx, char *ttly,int mtx,int mty,float x1,
   float y1,float deltax,float deltay);
void axis(float xpage,float ypage,char *ititl,int nchar,
   float axlen, float angle,float firstv,float deltav);
void curuxy(float *xx, float *yy, float x1, float y1, float deltax,
   float deltay,int nocx,int nocy,char *ic);
void factor (float fact) ;
void frame(void );
void gframe(int mode );
void gclip(char *cmd, float xlow,float ylow,float xhigh,float yhigh );
void gcont(float xx, float yy);
void gcursor(char *ctyp);
void gend(int mode);
void gfont(int ifont);
void ginfo(int *HasMouse, float *XminDev, float *YminDev,
   float *XmaxDev, float *YmaxDev, float *XminClip,
   float *YminClip, float *XmaxClip, float *YmaxClip,int *Color);
void ginitf(char *string, char *iconstr);
void gmesg(char *mesg);
void gmove(float xx, float yy);
void gread(char *fname, float X0, float Y0, float Xlow, float Ylow,
   float Xhigh, float Yhigh, int PageNum, float scalx, float scaly);
void gwrtxt(float xx,float yy,char *text,int flgbln);
void grdtxt(char *text,int lstr);
void gunit(char *str);
void gwidth(float width);
void line(float x[],float y[],int n,int inc,int lintyp,int inteq);
void lined(float x[],float y[],int n,int inc,int lintyp,
   int inteq,int ipat,float xlen);
void newpen(int j);
void number(float xpage,float ypage,float height, float fpn,
   float angle,int ndec);
void pend(void );
void pinit(void);
void pinitf(char *string);
void plot(float x,float y,int ipen);
void plotd(float xx,float yy,int ipat,float xlen);
void plots(int ibuf,int nbuf,int ldev);
void pltlgd(float *x,float *y,int n,float x1,float y1,
   float deltax,float deltay,int lintyp,int inteq, float ht,
   int nocx,int nocy,int ipat,float xlen);
void pltlog(float x[],float y[],int n,float x1,float y1,
   float deltax,float deltay,int lintyp,int inteq,
   float ht,int nocx,int nocy);
void pltscl(float x[],float axlen,int n,float *x1,
   float *deltax,int nocx);
void gscale(float x[],float xaxlen,int n,int inc);
```

```
void shadep(int narr, float *xarr, float *yarr);
void shader(float x1,float y1,float x2,float y2,int ipatx,
   int ipaty,float xlen,float ylen);
void shadet(float x1,float y1,float x2,float y2, float x3,
   float y3,int ipatx,int ipaty,float xlen,float ylen);
void shdsei(float x1,float y1,int ixy,int istnd,int iplmn);
void symbol (float xloc,float yloc,float height, char *inbuf,
   float angle,int nocar);
void where(float *xpag, float *ypag, float *fct);
/* interactive input */
void curixy(int *ix,int *iy,char *ic);
void curaxy(float *xx,float *yy,char *ic);
void currxy(float *xx,float *yy,char *ic);
void cross(int *ix,int *iy, char *s);
```

The following sections provide the detailed description of each routine.

---

algaxe

Use *algaxe* to draw logarithmic or linear axes at the current origin.

`algaxe(xaxlen,yaxlen,nocx,nocy,ttlx,ttly,mtx,mty,x1,y1,deltax,deltay)`

`float xaxlen;` *length of x-axis in user units (inch or cm). If <=0 do not plot*

float yaxlen;	*length of y-axis in user units (inch or cm). If <=0 do not plot*
float x1;	*First value in x-axis*
float y1;	*First value in y-axis*
float deltax;	*Number of x-units per user unit (inch or cm) if linear* *Length of cycle in user units (inch or cm) if logarithmic*
float deltay;	*Number of y-units per user unit (inch or cm) if linear* *Length of cycle in user units (inch or cm) if logarithmic*
char *ttlx;	*x-axis title*
char *ttly;	*x-axis title*
int mtx;	*Number of characters in x-axis title*
int mty;	*Number of characters in y-axis title*
int nocx;	*Number of logarithmic cycles on x-axis. If == 0, linear plot*
int nocy;	*Number of logarithmic cycles on y-axis. If == 0, linear plot*

See sample program *tstcur.c*

axis

Draw a linear axis scale.

`axis(xpage,ypage,title,nchar,axlen,angle,firstv,deltav)`

`float xpage;` *x-coordinate of starting point of axis in user units (inch or cm) relative to current origin*

float ypage; *y-coordinate of starting point of axis in user units (inch or cm) relative to current origin*

float axlen; *Length of axis in user units (inch or cm)*

float angle; *Angle (degrees) that axis makes with the x-axis*

float firstv; *Value of axis at (xpage, ypage)*

float deltav; *Change in axis value between tic marks per unit length (inch or cm). If deltav < 0, the value at (xpage, ypage) is a maximum, and values decrease along the axis*

char *title; *Axis title*

int nchar; *Number of characters in title. >0 for tic marks, numbers and title on counterclockwise side. <0 for tic marks, numbers and title on clockwise side.*

See the program example *line.c*.

curaxy

For terminals with cursor input, turn the cursor on, and acquire the coordinates of the cursor when any character is typed on the terminal. Return this character to the calling program.

`curaxy(xx,yy,ic)`

`float *xx;` *x-coordinate of cursor in user coordinates*

float *yy; *y-coordinate of cursor in user coordinates*

char *ic; *character returned with cursor coordinates. Must be defined as char ic[2];*

If this is used with a library that does not support interactive cursor input, then the values returned are *xx = 0.0, yy = 0.0,* and *ic = " ".* See sample program *tstcrs.c*

curixy

For terminals with cursor input, turn the cursor on, and acquire the coordinates of the cursor when any character is typed on the terminal. Return this character to the calling program. The coordinates returned are in the pseudoscreen coordinates relative to an origin at the lower left corner. The coordinates are effectively 0.001 inches.

`curixy(ix,iy,ic)`

`int *ix;`	*x-coordinate of cursor in plot coordinates*
int *iy;	*y-coordinate of cursor in plot coordinates*
char *ic;	*character returned with cursor coordinates. Must be defined as char ic[2];*

If this is used with a library that does not support interactive cursor input, then the values returned are *ix = 0*, *iy = 0*, and *ic = " "*. This is not too useful for the user, but this primitive is required to support the other three cursor input routines. See sample program *tstcrs.c*

currxy

For terminals with cursor input, turn the cursor on, and acquire the coordinates of the cursor when any character is typed on the terminal. Return this character to the calling program. The coordinates returned are user coordinates relative to the current origin.

`currxy(xx,yy,ic)`

`float *xx;`	*x-coordinate of cursor in user coordinates*
float *yy;	*y-coordinate of cursor in user coordinates*
char *ic;	*character returned with cursor coordinates. Must be defined as char ic[2];*

If this is used with a library that does not support interactive cursor input, then the values returned are *xx = 0.0*, *yy = 0.0*, and *ic = " "*. See sample program stcrs.c

curuxy

This procedure is used for interactive cursor graphics. By defining the parameters and drawing with algaxe, pltlog, pltscl and pltlgd, the screen coordinates are converted to user coordinates.

`curuxy(xx,yy,x1,y1,deltax,deltay,nocx,nocy,ic)`

`float *xx;`	*x-coordinate of cursor in user coordinates*
float *yy;	*y-coordinate of cursor in user coordinates*
float x1;	*User value of left side of x-axis*

float y1;	*User value of bottom side of y-axis*
float deltax;	*Number of x-units per user unit (inch or cm) if linear*
	Length of cycle in user units (inch or cm) if logarithmic
float deltay;	*Number of y-units per user unit (inch or cm) if linear*
	Length of cycle in user units (inch or cm) if logarithmic
int nocx;	*Number of logarithmic cycles on x-axis. If == 0, linear plot*
int nocy;	*Number of logarithmic cycles on y-axis. If == 0, linear plot*
char *ic;	*character returned with cursor coordinates. Must be defined as char ic[2];*

If this is used with a library that does not support interactive cursor input, then the values returned are *xx = 0.0, yy = 0.0,* and *ic = " ".* See sample program *tstcur.c*

---

factor

Use *factor* to scale all plot moves by *fact. fact < 1.0* **makes the plot uniformly smaller, while** *plot > 1.0* makes the plot uniformly larger than normal.

```
factor (fact)
float fact;
```
Multiply all moves by fact

---

frame

Start plot on a new page. On hardcopy output, the paper is advanced. On a graphics terminal, the display is erased.

```
frame()
```
See sample program *tabl.c*.

---

gclip

Define a clip region for plots.

```
void gclip(cmdstr, xlow, ylow, xhigh, yhigh)
char *cmdstr;
float xlow, xhigh, ylow, yhigh;
```
Define a clipping rectangle bounded by the coordinates (xlow,ylow) (xhigh, yhigh) if cmdstr = "ON" or remove all clip regions is cmdstr = "OFF" See sample programs *button.c*, *new.c* or *stars.c*.

gcont

Draw a line in the current color from the current point to the coordinates (xx,yy) Start plot on a new page. On hardcopy output, the paper is advanced. On a graphics terminal, the display is erased.

```
gcont(xx,yy)
real xx, yy
```

gcursor

Define the cursor to be used for interactive programs. The cursor can either be a hardware cursor, or an XOR cursor. The hardware cursor will always be seen, but the XOR color may be difficult to see with some colors.

Although hardware cursor if preferred since it can be easily seen against all backgrounds, interesting effects can be created if a different cursor is combined with the rectabgulathr region defined by the **`gclip()`** call. In this case the hardware cursor appears outside the rectangular clip region and the software cursor within.

```
gcursor(cursorstr)
char *cursorstr;
```

The character string defines the cursor type:

"Arrow"	Hardware arrow cursor
"Xorarrow"	XOR arrow cursor
"Crosshair"	XOR crosshair
"Plus"	XOR plus
"Box"	Draw a box (see cdraw.c example)
"Rubber"	Draw rubber band line (see cdraw.c example)
"Off"	Return to previous cursor (this only works back to one level)

The sample program *new.c* exercises the cursor. It also demonstrates how the cursor works when a rectangular clip region is defined. For example, one can have a crosshairs within a rectangular region but the cursor reverts to the hardware arrow cursor outside this region.

gframe

This is an extension of the frame() command. Normally a call for a page erase is only performed after the user manually clicks the mouse or hits the *Next* button at the top of the *plotmsw* or *plotxvig* presentation. The *gframe* permits an automatic move to the next page. The programmer can construct a button which is used to procede to the next page.

```
call gframe(mode)
integer mode
```

If *mode = 1*, automatically go to the next page, else wait for user interaction.
If *mode = 2*, if the screen was resized, redraw the window to the new CALPLOT 10x8 size, and then automatically go to the next page.

gend

Gracefully terminate the plotting session. this is an extension of *pend()* which is now *gend(0)*. For interactive graphics drivers, a *gend(1)* forces a termination of the graphics window. The use will not have to manually hit the **`Quit`** button.

```
gend(int mode)
```

This is meant to replace the old *plot(0.0,0.0,999)*.

gfont

Change the font in use by the *symbol()* call. By definition there is no upper limit to the number of fonts used. All plot drivers support the following. See the documentation for the specific plotting device for more detail.

Fonts Supported

Font Number	Font
0	Hardware default or override
1	Regular
2	Italic
3	Bold
4	Greek

The sample program *tabl.c* makes use of the *gfont(0)* call to permit a quick perusal of all fonts at plot time. For example if the mapping of symbol number is desired in Helvetica-

Bold, one plots the plot file generated by this program, *TABL* on the PostScript printer by the command

plotnps -F7 < TABL > PostScript_Language_File

```
gfont(ifont)
int ifont;   Font value
```

For generality, a font value of 9 would reduce to a value of 1 on a device that supports only the four fonts. Thus, fonts 2, 6, 10, etc., always refer to an italic font, and 3, 7, 11, etc., always will give a bold font. See the test program *gwid.c*.

---

ginfo

Obtain information about the graphics state. this information can be use to develop alternative presentations according to available screen area. For example, oen may use hardware fonts if there is enough space, but revert to simple vector fonts for very small work areas.

```
ginfo(HasMouse, XminDev, YminDev, XmaxDev, YmaxDev,
     XminClip, YminClip, Xmaxclip, YmaxClip, Color)
int *HasMouse.
float *XminDev, *YminDev, *XmaxDev, *YmaxDev;
float *XminClip, *YminClip, *Xmaxclip, *YmaxClip;
int *Color;
```

HasMouse	1 for true, 0 for false
XminDev, YminDev XmaxDev, YmaxDev	Display size in pixels
XminCLip, YminClip XmaxClip, YmaxClip	Current clip region in user units
Color	

See sample program *tabl.c*.

---

ginitf

Generalization of *pinitf*. Initialize plot. For non-interactive displays, a binary graphics meta-file which is used as input to the **calplot** commands. On an graphics terminal, this will initiate the graphics display. Up to the invocation of this call, the user may use C language IO to the screen. But once graphics are initialized, the user must use the *gwrtxt()* and *grdtxt()* to avoid any interference with the display.

When a plot meta-file is created, instead of creating the default plotXXXXX file as in *pinit()*, the user may control the naming and the redirection of output.

```
ginitf(plotstr,iconstr)
char *plotstr;      Name of plot file
```

If plotstr = "plot", open a unique file named plotXXXXX
If plotstr = "stdout", write the plot command stream to the standard output for redirection or piping into the calplot *command*
If plotstr = "INTER" enter interactive mode if supported by library
If plotstr = "anything else", the output will be placed in a file named "anything else".

*char *iconstr; name of title to be placed on the window manager frame, or when iconified.*

An interesting aspect is that one can have an interactive plot, then redirect the graphics to an output file, and then return to the interactive plot. This is indicated in the test program *button.c*.

iconstr is the name that should appear on the taskbar of the window in an interactive program. This is useful if the window is minimized so that the program can be restarted from the taskbar See sample program *button.c*.

---

gmesg

Put an informative message on the top bar of the X11 or Windows display. This is very useful in an interactive program for presenting information or program status.

```
gmesg(mesgstr)
char *mesgstr;
```

See sample program *new.c* or *clip.c*.

---

gmove

Lift the pen and move to the coordinates (xx,yy)

```
gmove(xx,yy)
real xx, yy
```

gread

Start plot on a new page. On hardcopy output, the paper is advanced. On a graphics terminal, the display is erased.

```
gread(fnamestr, X0, Y0, Xlow, Ylow, Xhigh, Yhigh,
     PageNum, scalx, Scaly)
char *fnamestr;
float X0, Y0;
float Xlow, Ylow, Xhigh, Yhigh;
int PageNum;
float scalx, scaly;
```

See sample program *gread.c* and *cdraw,c*.

grdtxt

Read a text string from the screen while in graphics mode. The input string is echoed, and a user defined erase key erases errors (For the PC this is a BackSpace). The input string is read until either a RETURN is entered, or until the input string length is exceeded.

For safety with any input, the string should be blank filled prior to input to remove the dregs of previous input.

```
grdtxt(text,lstr)
char *text;   String to be returned to program
int lstr;     Length of string
```

One in general can not use the *strlen()* routine to define the string, since it only reads up to the null character. This is best safely set up using macros. See sample program *gphtxt.c*.

```
#define NCHAR 10
char string[NCHAR];
grdtxt(string,NCHAR);
```

gunit

All plot movements after this point will be in the units defined. Initially all movements are in inches.

```
gunit(str)
char *str;   Either the strings "CM", "cm", "in" or "IN"
```

```
gunit("in");/* draw a square */
plot(0.0,0.0,-3);
plot(1.0,0.0,2);
plot(1.0,1.0,2);
gunit("cm");
plot(0.0,2.54,2);
plot(0.0,0.0,2);
```

gwidth

Make the width of all plotted lines *width* user units.

```
gwidth(width)
float width;Width of all lines drawn in user units
```

A value of *width = 0.0* will be interpreted as the minimally resolvable width, and not as a blank line. Since drawing a wide line takes time on most devices, the plotdrivers have an option to turn off linewidth. See sample program *gwid.c*.

gwrtxt

Write graphic text to the screen, using hardware character capability if possible. The text is displayed horizontally parallel to the x-axis.

```
gwrtxt(xx,yy,text,flgbln)
float xx;    (xx,yy) are coordinates of lower left corner of string
float yy;
char *text;  Text to be displayed, NULL terminated
int flgbln;  0 - write text out
             1 - overwrite existing output with blanks prior to write,
             filling with the background color of the display
             2 - display output in reversed video mode
```

For almost all displays, no more than 80 characters can be displayed horizontally.

The text is written in using the current pen color defined by the last call to *newpen()* with

argument < 1000. In the reversed video mode, the default newpen(1) color will define the background in the display box, and the normal background will be used for the text. In general, this will yield a white writing on a black background for laser or printer output. See test program *gphtxt*.

---

line

Draw a line, optionally using symbols to represents data points.

`line(x,y,n,inc,lintyp,inteq)`

`float x[];`	*Array of abscissa*
float y[];	*Array of ordinates*
int n;	*Number of points in array to be plotted*
int inc;	*plot every* inc'th *point*
	there are n*inc + inc + 1 *points in the array*
int lintyp;	*!=0 plot symbol* inteq*at each abs*(lintyp) *point*
	=0 plot line only
	<0 plot only symbols, no connecting line
	>0 plot symbols, with connecting line
int inteq;	*If* lintyp != 0, *plot this symbol*

The mapping of the value of *inteq* to a specific symbol is shown by the test program *tabl.c*.

The arrays *x[]* and *y[]* must have the *firstv* and *deltav* entries placed into them by the *gscale()* routine. Note also the need for additional array storage beyond *n*. See test program *testl.c*.

---

lined

`lined(x,y,n,inc,lintyp,inteq,ipat,xlen)`

`float x[];`	*Array of abscissa*
float y[];	*Array of ordinates*
int n;	*Number of points in array to be plotted*
int inc;	*plot every* inc'th *point*
	there are n*inc + inc + 1 *points in the array*
int lintyp;	*!=0 plot symbol* inteq*at each abs*(lintyp) *point*
	=0 plot line only
	<0 plot only symbols, no connecting line
	>0 plot symbols, with connecting line
int inteq;	*If* lintyp != 0, *plot this symbol*
int ipat;	*Binary pattern for display (0,31)*

float xlen; *Length in user units of each bit of the pattern*

The mapping of the value of *inteq* to a specific symbol is shown by the test program *tabl.c*.

The arrays *x[]* and *y[]* must have the *firstv* and *deltav* entries placed into them by the *gscale()* routine. Note also the need for additional array storage beyond *n*. See test program *testld.c*.

---

newpen

Select pen color for the plot.

newpen(ipen)

int ipen; *Pen color*

The mapping of colors is hardware dependent. *ipen=0* indicates the background color, which is not useful, unless one wishes to overwrite previous out, especially with shading. *0 < ipen < 1000* indicates a color. Hardware devices with jsut 4 pen colors, will use a mod operator to map, *ipen=9* into the color for *ipen=1*. The values *1000 <= ipen <= 1100* are used to define halftone or color fill patterns for the *shader()* and *shadet()*. See sample program *plttst.c* for color selection. See sample programs *gryscl.c*, *grytst.c*, or *nseitst.c* for colored (halftoned) shading.

---

number

Convert a floating point number to a character string, and plot the string (number).

number(xpage,ypage,height,fpn,angle,ndec)

float xpage; *x-coordinate of first lower left corner of number*
xpage = 999.0 *continues with* xpage *set to the*
final x-coordinate from the last call to symbol or number

float ypage; *y-coordinate of first lower left corner of number*
ypage = 999.0 *continues with* ypage *set to the*
final y-coordinate from the last call to symbol or number

float height; *Height of each character plotted*

float fpn; *Number to be plotted*

float angle; *Plot number at* angle *degrees with respect to x-axis*

int ndec; *Numerical precision of number*
ndec - *number of figures to the right of the decimal place*
If ndec = -1, *the floating point number will appear as*
an integer. No decimal place is plotted.

If ndec >= 1000, *an exponential FORTRAN format is invoked*

of the form Ew.d. *The precision* d *is set by defining*
ndec = 1000 + d. *Note that all numbers are left justified.*
For this exponential format, the total width is (d + 6) * height**.**

If ndec >= 2000 and <= 2009, *Scientific notation is used*
The mantissa has a precision given by d, *set by defining*
ndec = 2000 + d. *Note that all numbers are left justified.*
For this exponential format, the total width about (d + 6)*height**.**

The use of *ndec >=1000* is an extension of the original definition made for these libraries.

The convention for character size, is that the *height* be the distance from the baseline to the maximum symbol elevation, for example the distance from the base to the top if the letters **I** or **A**. The convention for the intercharacter spacing is that the character with be equal to the height, with the actual character filling *0.6 * height* of this space. Thus the lower left corners of the two **2**'s in **22** are separated by *height* units, and the space between the two characters is the *0.4 * height*.

See sample program *tabl.c*.

---

pend

Gracefully terminate the plotting session.

```
pend()
```

This is meant to replace the old *plot(0.0,0.0,999)*.

---

pinit

Initialize plot. For non-interactive displays, a **plotXXXXX** file which is used as input to the **calplot** commands. On an graphics terminal, this will initiate the graphics display. Up to the invocation of this call, the user may use C language IO to the screen. But once graphics are initialized, the user must use the *gwrtxt()* and *grdtxt()* to avoid any interference with the display.

```
pinit()
```

pinitf

Open the Initialize plot. For non-interactive displays, a binary graphics meta-file which is used as input to the **calplot** commands. On an graphics terminal, this will initiate the graphics display. Up to the invocation of this call, the user may use C language IO to the screen. But once graphics are initialized, the user must use the *gwrtxt()* and *grdtxt()* to avoid any interference with the display.

When a plot meta-file is created, instead of creating the default plotXXXXX file as in *pinit()*, the user may control the naming and the redirection of output.

```
pinitf(string)
char *string;
```
Name of plot file
If string = "plot", open a unique file named plotXXXXX
If string = "stdout", write the plot command stream to the standard output for redirection or piping into the calplot *command*
If string = "anything else", the output will be placed in a file named "anything else".

All test programs direct output to a file for that program, e.g., *tabl.c* creates a plot file *TABL*.

plot

Move the pen from the current position to the given coordinates.

```
plot(x,y,ipen)
float x;
float y;
int ipen;
```
(x,y) *are the desired position*
-3 - pen up during move (hidden vector), establish new origin at end of movement
-2 - pen down during move (drawn vector), establish new origin at end of movement
+2 - pen down during move (drawn vector), do not establish new origin at end of movement
+3 - pen up during move (drawn vector), do not establish new origin at end of movement
999 - terminate plot after the move. For compatibility only, use pend()

plotd

This connects the current point and the point *(xpage, ypage)* with a dashed line given by the pattern *ipat*. The attached example tests this routine. The idea is that the integer *ipat* is a number between 0 and 31 which defines a five bit pattern, each of which represents a pen movement of *xlen* units. As each segment is plotted, the next bit in *ipat* is interrogated whether it is 0 or 1. A 1 represents a plotted vector, and a zero, an unplotted vector. Because a sixth bit is implicitly assumed to be zero, certain patterns are just a linear shift of others, e.g., 1 and 2, 1 and 4, etc. The unique patterns available have *ipat* values of *0, 1, 3, 5, 7, 9, 11, 13, 15, 21, 23, 27, and 31*. The attached figure shows this correspondence.

```
plotd(xx,yy,ipat,xlen)
```

float xx;	*x-coordinate at end of line*
float yy;	*y-coordinate at end of line*
float xlen;	*Length in user units of each bit of the pattern*
int ipat;	*Binary pattern for display (0,31)*

See sample program *dtest.c*.

plots

Initialize plot stream. The use of *pinitf* or *ginitf* is preferred.

```
plots(ibuf,nbuf,ldev)
int ibuf;
int nbuf;
int ldev;
```

This is used for upward compatibility of old programs. This routine just invokes *pinit()*.

pltlgd

A generalization of the *line()* routine, except that logarithmic scaling is supported. In addition, dashed lines are permitted. It differs from the *line()* routine in that each point is plotted, and that the coordinate arrays need not have scaling numbers put into them.

```
pltlgd(x,y,n,x1,y1,deltax,deltay,lintyp,inteq,ht,nocx,nocy,ipat,xlen)
```

float x[];	*Array of abscissa to be plotted*
float y[];	*Array of ordinates to be plotted*
int n;	*Number of points to be plotted*
float x1;	*If linear axis, this is the value at the left end of the x-axis. If logarithmic, this is the* log10 *of the*

left end of the x-axis. This will be the exponent displayed.

float y1; *If linear axis, this is the value at the left end of the y-axis. If logarithmic, this is the* log10 *of the left end of the y-axis. This will be the exponent displayed.*

float deltax; *If linear, this is the number of units per basic unit (inch or cm) along the x-axis. If logarithmic, this is the number of basic units(inch or cm) per cycle of the logarithmic axis.*

float deltay; *If linear, this is the number of units per basic unit (inch or cm) along the y-axis. If logarithmic, this is the number of basic units(inch or cm) per cycle of the logarithmic axis.*

int lintyp; *!=0 plot symbol* inteq*at each abs*(lintyp) *point*
=0 plot line only
<0 plot only symbols, no connecting line
>0 plot symbols, with connecting line

int inteq; *If* lintyp != 0, *plot this symbol*

float ht; *Height of symbol plotted on line*

int nocx; *0 - x-axis is linear*
>0 - x-axis is logarithmic with this many cycles

int nocy; *0 - y-axis is linear*
>0 - y-axis is logarithmic with this many cycles

int ipat; *Binary pattern for display (0,31)*

float xlen; *Length in user units of each bit of the pattern*

See the sample program *tstcur.c*.

---

pltlog

A generalization of the *line()* routine, except that logarithmic scaling is supported. It differs from the *line()* routine in that each point is plotted, and that the coordinate arrays need not have scaling numbers put into them.

`pltlog(x,y,n,x1,y1,deltax,deltay,lintyp,inteq,ht,nocx,nocy)`

`float x[];` *Array of abscissa to be plotted*

float y[]; *Array of ordinates to be plotted*

int n; *Number of points to be plotted*

float x1; *If linear axis, this is the value at the left end of the x-axis. If logarithmic, this is the* log10 *of the left end of the x-axis. This will be the exponent displayed.*

float y1; *If linear axis, this is the value at the left end of the y-axis. If logarithmic, this is the* log10 *of the left end of the y-axis. This will be the exponent displayed.*

float deltax; *If linear, this is the number of units per basic unit (inch or cm) along the x-axis. If logarithmic, this is the number of basic units(inch or cm) per cycle of the logarithmic axis.*

float deltay; *If linear, this is the number of units per basic unit (inch or cm) along the y-axis. If logarithmic, this is the number*

of basic units(inch or cm) per cycle of the logarithmic axis.

int lintyp; *!=0 plot symbol* inteq*at each abs*(lintyp) *point*
=0 plot line only
<0 plot only symbols, no connecting line
>0 plot symbols, with connecting line

int inteq; *If* lintyp != 0, *plot this symbol*

float ht; *Height of symbol plotted on line*

int nocx; *0 - x-axis is linear*
>0 - x-axis is logarithmic with this many cycles

int nocy; *0 - y-axis is linear*
>0 - y-axis is logarithmic with this many cycles

See sample program *tstcur.c*.

---

pltscl

A generalization of the *call gscale* call to include establishing scaling for logarithmic plots. If *nocx* is zero, linear scaling is performed by a call to the subroutine *gscale*, while if *nocx* is greater than 0, logarithmic scaling is performed to make a integral number of cycles. The plot is scaled to include the largest data value, lower values may not be plotted.

`pltscl(x,axlen,n,x1,deltax,nocx)`

`float x[];` *Array to be scaled to fit axis length*

float axlen; *Length of axis*

int n; *Number of points in the array*

float *x1; *A value returned by the program. For a linear plot, this gives the user value at the lowest value of th axis, while for logarithmic plots this is the* log10 *of the lowest value.*

float *deltax; *Conversion from user values to user coordinates. For a linear plot, this is the number of user values per basic user coordinate (cm or inch). For a logarithmic plot, this is the number of basic user units (cm or inch) per cycle.*

int nocx; *> 0 - number of logarithmic cycles along the axis*
0 - axis is linear

{ *> 0 - number of logarithmic cycles along the axis*
0 - axis is linear

See sample program *tstcur.c*.

gscale

This is used for automatic scaling of plot axes and for line call. array[i] is examined to choose a nice first value, firstv above, which is returned as array[npts] and a reasonable increment, deltav, which is returned as array[npts+abs(inc)]. deltav is the number of array[i] units per inch. To obtain the plot space coordinates, programs compute actual movement = (array[i] - firstv)/deltav.

This is used in conjunction with *line()*.

```
gscale(x,xaxlen,n,inc)
```

float x[];	*Array to be scaled*
float xaxlen;	*Length of axis on which array is to fit*
int n;	*Number of points in the array*
int inc;	*Increment for plotting, e.g., plot each abs(*inc*) point. A negative value indicates that the computed* firstv *should be a maximum rather than a minimum.*

See sample program *line.c*.

shadep

Shade a polygon with the current pen color

```
shadep(narr, xarr, yarr)
integer narr
real xarr(*), yarr(*)
```

See sample program *new.c*, *stars.c* and *grid.c*.

shader

This shades a rectangular area using different patterns in the x- and y-directions. The meaning of the pattern is described under *plotd*. The only real difference is that this pattern is generated in the device filter and not in the source level for efficiency. Thus this differs from *plotd* and *lined* in that the pattern is repeated with respect to the absolute page coordinate, hence the pattern within a rectangular region is the same, but shifted, when the rectangle is placed at different positions on the page. You may imagine this as having a uniform background pattern, and moving a window across.

```
shader(x1,y1,x2,y2,ipatx,ipaty,xlen,ylen)
```

float x1;	(x1,y1) *are one corner of the rectangle*
float y1;	

float x2;	(x2,y2) *are opposite corner of the rectangle*
float y2;	
int ipatx;	*bit pattern along the x-axis*
int ipaty;	*bit pattern along the y-axis*
float xlen;	*length of each bit of the* ipatx *pattern along the x-axis*
float ylen;	*length of each bit of the* ipaty *pattern along the y-axis*

If *ipatx = 0* and *ipaty = 0*, then a solid fill is made using the the current shading value defined by *newpen()*. See the test programs *pattst.c*, *gryscl.c*, and *grytst.c*.

---

shadet

This shades a triangular area using different patters in the x- and y- directions. The pattern plotted is defined under the description of plotd(). The only minor problem is that the plot pattern is generated in the hardware specific plot filter, and not in the source level for efficiency. Thus this differs from plotd() and lined() in that the pattern is defined with respect to the absolute page coordinate, hence the pattern within the triangular region is shifted within the triangle, when the triangle is moved to different positions on the page. The underlying pattern is fixed. In addition, the agreement of this pattern on different devices will not be perfect, due to different hardware resolution.

`shadet(x1,y1,x2,y2,x3,y3,ipatx,ipaty,xlen,ylen)`

`float x1;`	(x1,y1) *are one corner of the triangle*
float y1;	
float x2;	(x2,y2) *are another corner of the triangle*
float y2;	
float x3;	(x2,y2) *are third corner of the triangle*
float y3;	
int ipatx;	*bit pattern along the x-axis*
int ipaty;	*bit pattern along the y-axis*
float xlen;	*length of each bit of the* ipatx *pattern along the x-axis*
float ylen;	*length of each bit of the* ipaty *pattern along the y-axis*

If *ipatx = 0* and *ipaty = 0*, then a solid fill is made using the the current shading value defined by *newpen()*. See the test program *tritst.c* for usage. In addition see the test programs *pattst.c* for more patterns and *dtest.c* for the bit patterns.

shdsei

This is a special routine designed to invoke shaded area plots of seismic traces, in a manner that is very efficient at the source level. The idea used is that this routine be invoked twice for each trace plotted, once to turn the shading on, and then after the trace is plotted, to turn the shading off. The center line of the trace must be defined in plot space coordinates. The hardware specific plot filter will use this information to shade the trace as it is plotted. The time axis of the trace can be parallel to either the x-axis or the y-axis. Positive or negative amplitude fill can be done.

`shdsei(x1,y1,ixy,istnd,iplmn)`

`float x1;` *x-coordinate of the first point of the trace, assuming that the first time sample has zero amplitude.*

float y1; *y-coordinate of the first point of the trace, assuming that the first time sample has zero amplitude. If the trace is plotted parallel to the x-axis, then only the* y1 *value is meaningful since this states where the amplitude = 0 line is.*

int ixy; *Flag to indicate whether the time axis is parallel to the x-axis* (0) *or the y-axis* (1)**.**

int istnd; *Flag to indicate whether to turn strace shading on* (1) *or off* (0)**.**

int iplmn; *(Flag to indicate whether positive amplitudes are to be shaded* (0) *or negative amplitudes* (1) *or both* (2)*. If trace is plotted parallel to x-axis, then positive amplitudes are y-values* > y1**.**

See the test programs *seitst.c* for simple usage and *nseitst.c* for use together with shading to display trace attributes in addition.

symbol

Use symbol to plot a character or special symbol at the current position. Two different syntaxes are used, according to whether a string is plotted or just a single character or special symbol.

`symbol (xpage,ypage,height,inbuf,angle,nchar)`

`float xpage;` *x-coordinate of first lower left corner of number* xpage = 999.0 *continues with* xpage *set to the final x-coordinate from the last call to symbol or number*

float ypage; *y-coordinate of first lower left corner of number* ypage = 999.0 *continues with* ypage *set to the final y-coordinate from the last call to symbol or number*

float height; *Height of each character plotted*

float angle; *Plot string at* angle *degrees with respect to x-axis*
char *inbuf; *Character string to be plotted*
int nchar; *If* nchar > 0 *plot* nchar *characters of the string*
If nchar < 0 *plot only a single symbol defined*
by the number stored in inbuf[0]. *If* nchar = -1,
no line is drawn from the last position to this coordinate.
If nchar < -1, *then a connecting line is drawn.*
Finally, if nchar < 1, *the first 16 symbols will be*
centered at the coordinates (xpage, ypage).

The convention for character size, is that the *height* be the distance from the baseline to the maximum symbol elevation, for example the distance from the base to the top if the letters **I** or **A**. The convention for the intercharacter spacing is that the character with be equal to the height, with the actual character filling *0.6 * height* of this space. Thus the lower left corners of the two **A**'s in **AA** are separated by *height* units, and the space between the two characters is the *0.4 * height*. This information is given for positioning of characters. See the test program *tabl.c* for the mapping between *inbuf* and a specific symbol for negative values of *nchar*..

```
symbol(1.0,1.0,0.20,"HELLO WORLD",0.0,11);
```
plots string parallel to x-axis starting at (1.0,1.0)

```
symbol(2.0,2.0,0.10,"0",0.0,-1);
```
centers a square symbol at position (2.0,2.0)

---

where

Determine the current plotting point position in user coordinates.

```
where(xpage, ypage, fct)
float *xpage;
```
x - coordinate of current position
float *ypage; *y - coordinate of current position*
float *fct; *A necessary parameter for upward compatibility*
Always returned as 1.0

3. Internal Routines

The following are internal routines used by these libraries. The user need only note that these names are already taken, and that a user program must avoid defining these names or, in general, using these names directly.

This section contains auxiliary routines and data structures required to implement the *CALPLOT* package. For the most part, users must not redefine any of these names or attempt to use them. Otherwise *CALPLOT* will not function as desired. The number of externally defined variables that may be inadvertently accessed is large for the interactive

versions of this package. Thus the user is cautioned to be very careful when using the C **extern** directive. Except for the **`gintxt()`**, **`gomesg()`**, **`gottxt()`**, these reserved routines and variables befin with the prefixes *dc_*, *dp_*, *dv_*, *di_* or *do_*.

```
#ifndef INT
#ifdef MSDOS
#define INT long
#else
#define INT int
#endif
#endif

extern double dc_Nxpi;
extern double dc_Nypi;
extern double dc_scalefac;
extern double dc_xoff;
extern double dc_xscale;
extern double dc_yoff;
extern double dc_yscale;
extern INT dc_ClipRight, dc_ClipTop, dc_ClipBottom, dc_ClipLeft;
extern INT dc_ClipRight_sv, dc_ClipTop_sv, dc_ClipBottom_sv, dc_ClipLeft_sv;
extern INT dc_color;
extern INT dc_ColorInfo;
extern INT dc_curfont;
extern INT dc_curpen;
extern INT dc_hardwarefill;
extern int dc_hasmouse;
extern INT dc_herelinewidth ;
extern INT dc_iseps;
extern INT dc_left, dc_bottom, dc_right, dc_top;
extern INT dc_linewidth;
extern INT dc_mapcurpen;
extern INT dc_newlinewidth;
extern INT dc_oldx1, dc_oldy1;
extern INT dc_rotate;
extern INT dc_sdx;
extern INT dc_shadeoff;
extern INT dc_shdcur;
extern INT dc_shdon;
extern INT dc_xlinewidth;
extern INT dc_ylinewidth;
extern INT dp_ClipRightSave, dp_ClipLeftSave,
   dp_ClipTopSave, dp_ClipBottomSave;
extern INT dv_lineclip(INT *x0,INT *z0,INT *x1,
   INT *z1,INT dc_left,INT dc_bottom,INT dc_right,INT dc_top);
int is_long;
extern struct Gcplot Gcpl;
extern void (*do_arc)(INT Xi,INT Yi,INT X0,INT Y0,INT X1,INT Y1);
```

```
extern void (*do_circle)(INT Xi,INT Yi,INT r);
extern void (*do_clip)(INT cmd, INT X0,INT Y0,INT X1,INT Y1);
extern void (*do_cont)(INT X0,INT Y0);
extern void (*do_cross)(int *X0,int *Y0, char *c);
extern void (*do_cursor)(INT curstyp);
extern void (*do_erase)(INT mode);
extern void (*do_fillp)(INT n,INT *x,INT *y);
extern void (*do_fillr)(INT X0,INT Y0,INT X1,INT Y1,
  INT patx,INT paty,INT lenx,INT leny);
void (*do_fills)(INT X0,INT Y0,INT ixy,INT istnd,INT iplmn);
extern void (*do_fillt)(INT X0,INT Y0,INT X1,INT Y1,INT X2,INT Y2,
  INT patx,INT paty,INT lenx,INT leny);
void (*do_font)(INT Xi);
extern void (*do_gintxt)(int cnt, char *s);
extern void (*do_gottxt)(char *s);
extern void (*do_gsymb)(INT X0,INT Y0,INT X1,INT Y1,INT n,char *s);
extern void (*do_gwid)(INT wid);
extern void (*do_info)(INT *HasMouse, INT *XminDev, INT *YminDev,
  INT *XmaxDev, INT *YmaxDev, INT *XminClip,
  INT *YminClip, INT *XmaxClip, INT *YmaxClip, INT *Color);
extern void (*do_label)(char *s);
extern void (*do_linec)(INT X0,INT Y0,INT X1,INT Y1);
extern void (*do_linemod)(char *s);
extern void (*do_move)(INT X0,INT Y0);
extern void (*do_pen)(INT Xi);
extern void (*do_point)(INT X0,INT Y0);
extern void (*do_space)(INT X0,INT Y0,INT X1,INT Y1);
extern void dv_clip();
extern void dv_closepl(void);
extern void dv_concur(INT isx,INT isy);
extern void dv_cursor(INT curstyp);
extern void dv_erase(INT mode);
extern void dv_fillp(INT n, INT *x, INT *y);
extern void dv_font(INT Xi);
extern void dv_movcur(INT isx,INT isy);
extern void dv_openpl(void);
extern void dv_pen(INT xi);
extern void dv_rline(INT X0,INT Y0,INT X1,INT Y1);
extern void dv_rliner(INT x0,INT z0,INT x1,INT z1);
extern void dv_zzpoint(INT x, INT y);
int     dc_shdse;
void di_arc(INT xi,INT yi,INT X0,INT Y0,INT X1,INT Y1);
void di_circle(INT xi,INT yi,INT r);
void di_circle( INT x, INT y, INT r);
void di_clip(INT cmd, INT X0,INT Y0,INT X1,INT Y1);
void di_closepl(void);
void di_cont(INT X0,INT Y0);
```

```
void di_cross(INT *ix, INT *iy, char *c);
void di_cursor(INT curstyp);
void di_erase(INT mode);
void di_fillp(INT n, INT *x, INT *y);
void di_fillr(INT X0,INT Y0,INT X1,INT Y1,
  INT patx,INT paty,INT lenx,INT leny);
void di_fills(INT X0,INT Y0,INT ixy,INT istnd,INT iplmn);
void di_fillt( INT X0,INT Y0,INT X1,INT Y1,INT X2,INT y2,
  INT patx,INT paty,INT lenx,INT leny);
void di_font(INT font);
void di_gintxt(int cnt,char *s);
void di_gottxt(char *s);
void di_gsymb(INT x,INT y,INT ht,INT ang,INT nchar,char *str);
void di_gwid(INT width);
void di_info(INT *HasMouse, INT *XminDev, INT *YminDev,
  INT *XmaxDev, INT *YmaxDev, INT *XminClip,
  INT *YminClip, INT *XmaxClip, INT *YmaxClip, INT *Color);
void di_label(char* s);
void di_linec(INT X0,INT Y0,INT X1,INT Y1);
void di_linemod( char *s );
void di_move(INT xi,INT yi);
void di_openpl(void);
void di_pen(INT xi);
void di_point(INT X0,INT Y0);
void di_space(INT X0,INT Y0,INT X1,INT Y1);
void dp_arc(INT Xi,INT Yi,INT X0,INT Y0,INT X1,INT Y1);
void dp_circle(INT Xi,INT Yi,INT r);
void dp_clip(INT cmd, INT X0,INT Y0,INT X1,INT Y1);
void dp_cont(INT X0,INT Y0);
void dp_cross(INT *ix, INT *iy, char *c);
void dp_cursor(INT curstyp);
void dp_erase(INT mode);
void dp_fillp(INT n,INT *x,INT *y);
void dp_fillr(INT X0,INT Y0,INT X1,INT Y1,
void dp_fills(INT X0,INT Y0,INT ixy,INT istnd,INT iplmn);
void dp_fillt(INT X0,INT Y0,INT X1,INT Y1,INT X2,INT Y2,
  INT patx,INT paty,INT lenx,INT leny);
void dp_font(INT Xi);
void dp_gintxt(int  cnt, char *s);
void dp_gottxt(char *s);
void dp_gsymb(INT X0,INT Y0,INT X1,INT Y1,INT n,char *s);
void dp_gwid(INT wid);
void dp_info(INT *HasMouse, INT *XminDev, INT *YminDev,
  INT *XmaxDev, INT *YmaxDev, INT *XminClip,
  INT *YminClip, INT *XmaxClip, INT *YmaxClip, INT *Color);
void dp_label(char *s);
void dp_linec(INT X0,INT Y0,INT X1,INT Y1);
```

```
void dp_linemod(char *s);
void dp_move(INT X0,INT Y0);
void dp_pen(INT Xi);
void dp_point(INT X0,INT Y0);
void dp_space(INT X0,INT Y0,INT X1,INT Y1);
void dv_gread(int lf,char *fname, INT NumX, INT NumY, INT LowX,
  INT LowY, INT HighX, INT HighY, INT Num, INT Sclx, INT Scly);
void dv_symvec(INT xloc,INT yloc,INT ht,char *s,INT ang,INT nochar);
void dv_zpoint(INT x,INT y);
void gintxt(int cnt, char *s);
void gomesg(int cnt, char *mesg);
void gottxt(int cnt,char *s);
```

CHAPTER 4
COLORS AND FONTS

The proper use of color is a very important skill to present information. The use of color can be overdone and may not be appropriate for publication because of printing costs. The use of fonts can also be counter-productive if the font used distracts the reader from the information to be presented. This chapter addresses the current *CALPLOT* color and font model.

1. Color

CALPLOT was originally written for use with a single or just a few colors. Modern graphics displays can display 24-bit color to represent images, but this palette is usually not necessary for scientific graphics. *CALPLOT* supports a reduced palette of 100 colors, the representation of which is made at the time of invoking a device dependent plot filter program, such as **`plotxvig`**, **`plotmsw`**, **`plotnps`** or **`plotgif`**.

Color in defined in a program through the *newpen* call, either as

```
call newpen(kolor)
```

in FORTRAN or

```
newpen(kolor);
```

in C. The integer *kolor* can take on a number of values. If the values are between 1 and 999, the colors cycle between the display foreground, red, green blue,m orange, blue-green and yellow. If the values of *kolor* are between 1000 and 1100, the continuous palette between the colors represented by the indices 1000 and 1100 is sampled.

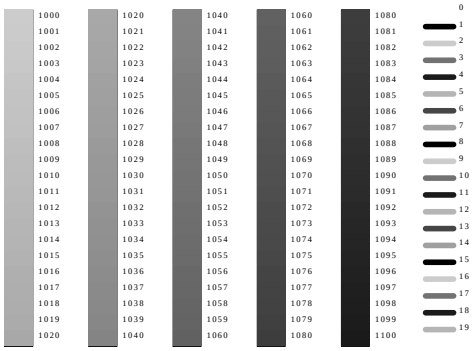
Since the actual color is a property of capabilities of the display device, we cannot be too specific here. The compter screen display programs, **`plotxvig`** and **`plotmsw`**, and the image file programs, **`plotgif`** and **`plotpng`**, permit an inversion of the foreground and background with a **`-I`** flag. Normally the background is white and the foreground color is black. If the **`-I`** flag is invoked, the background is black and the foreground is white. The PostScript driver, **`plotnps`** disbales the **`-I`** flag to permit only a white background. The **`plot4014`** only supports color as implemented in the **`TeraTerm`** Tektronix implementation. If **`plotxvig`** is used on older computers which support a limited color palette or only monochrome, the **`plotxvig`** will approximate the 100 different colors through dithering.

Four different color models can be invoked through the **-G**, **-K**, **-KR** and **-KB** flags. The **-G** flag is useful for publication. The **-KR** and **-KB** are useful for representing 2-D surfaces.

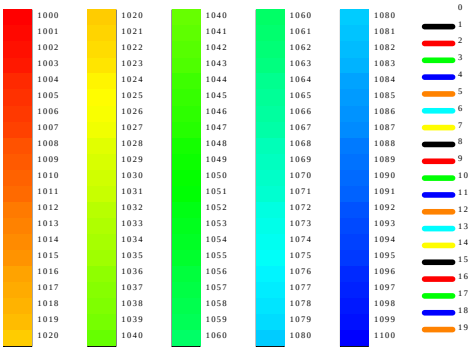
Table 4.1. Calplot colors

Kolor	-G	-K	-KR	-KB
0	Background	Background	Background	Background
1	Foreground	Foreground	Foreground	Foreground
2	(see below)	Red	(see below)	(see below)
3		Green		
4		Blue		
5		Orange		
6		Blue-Green		
7		Yellow		
8	Foreground	Foreground	Foreground	Foreground
9		Red		
999				
1000	Lt. Gray	Red	Red	Blue
1025		Orange	Light Red	Light Blue
1050	Med.Gray	Green	White	White
1075		BlueGreen	Light Blue	Light Red
1100	Dk. Gray	Blue	Blue	Red

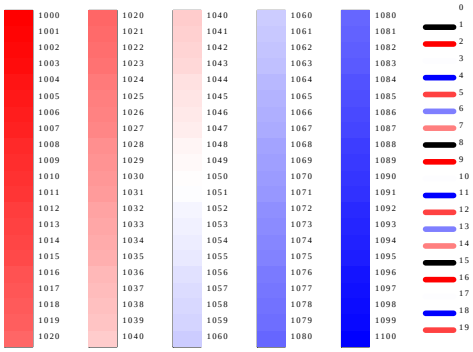
Normal plotting uses the **-G** and **-K** flags. Displays of continuous color maps can use the **-KR** and **-KB** modes if the color indices are programed to represent a range of negative - positive values with white representing a median value. The following figures show the resulting colors for a given choice of the `kolor` index.



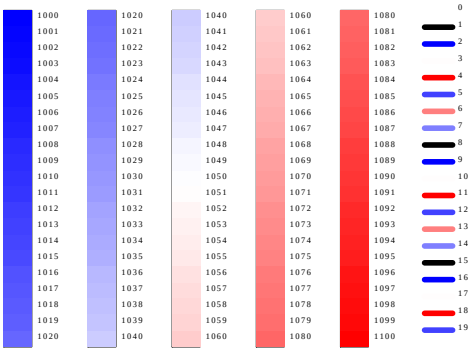
plotnps -G < GRAYSC



plotnps -K < GRAYSC



plotnps -KR < GRAYSC



plotnps -KB < GRAYSC

2. Fonts

CALPLOT invokes fonts through the *gfont* call, by

```
call gfont(font)
```

in FORTRAN or

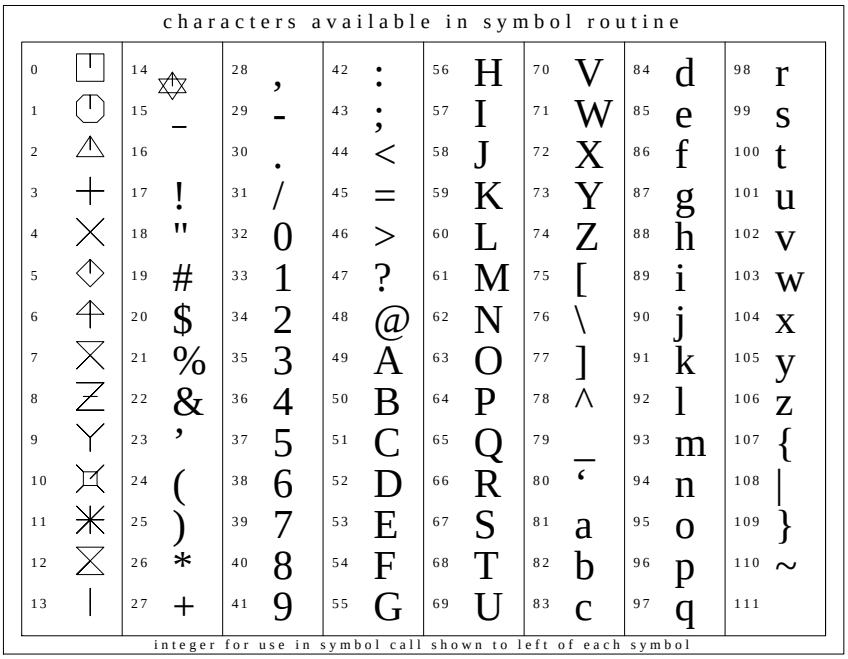
```
gfont(font);
```

in C. The integer *font* can take on a number of values. The interpretation is hardware dependent. The following table shows the relation between the value of the integer *font* and the resultant font on PostScript and displays wihc use the built in vector font. The value *font = 0* is very special, since the output can be overridden by invoking **`plotnps`**.

Table 4.2. CALPLOT fonts

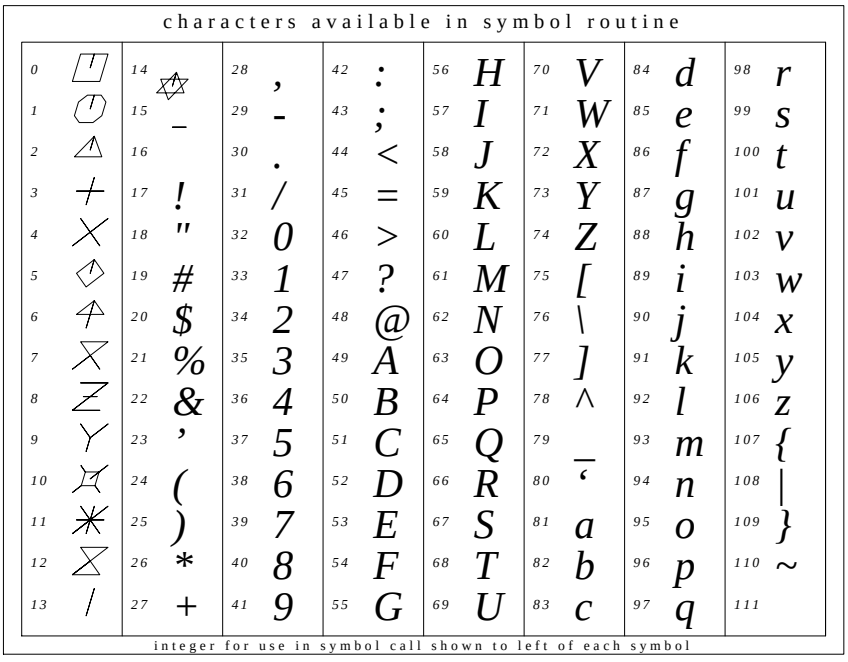
Font	Vector Font	**`PostScript`**
0	normal	Times-Roman but can be overridden using the -Ff flag
1	normal	Time-Roman
2	italic	Times-Italic
3	bold	Times-Bold
4	symbol(Greek)	Symbol(Greek)
5	normal	Helvetica-Roman
6	italic	Helvetica-Italic
7	bold	Helvetica-Bold
8	symbol(Greek)	Symbol(Greek)
9	normal	Courier-Roman
10	italic	Courier-Italic
11	bold	Courier-Bold
12	symbol(Greek)	Symbol(Greek)

To illustrate the fonts permissible with PostScript, the following 12 figures show the result of using the **`-Ff`** flag. I was able to do this since the command *gfont* was not used in the sample program *tabl*, so that the font selection is made a time of creation of the PostScript, in this case, and Encapsulated PostScript file. For publications plots, I normally use the Helvetica-Bold option.



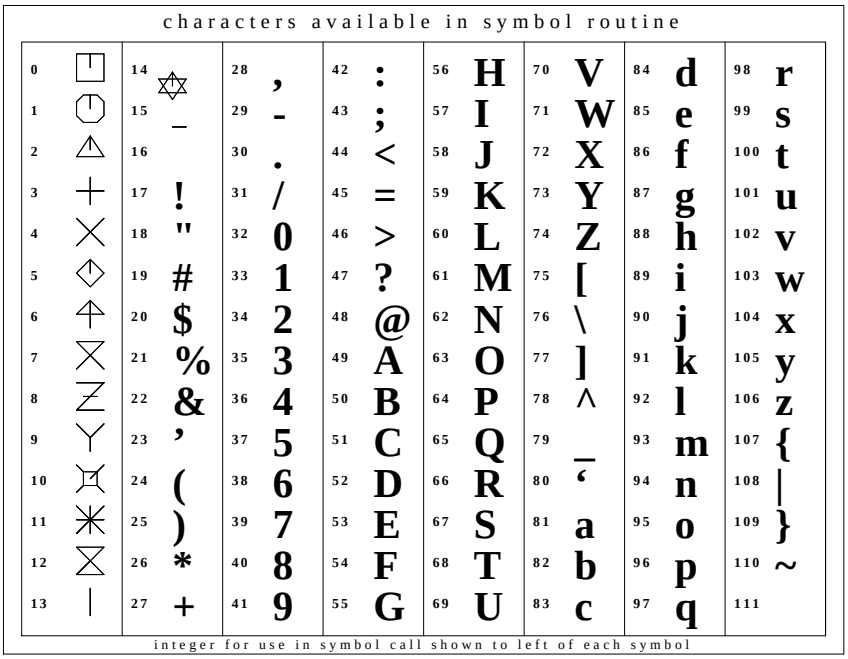
0	□	14	✡	28	,	42	:	56	H	70	V	84	d	98	r				
1	○	15	_	29	-	43	;	57	I	71	W	85	e	99	s				
2	△	16		30	.	44	<	58	J	72	X	86	f	100	t				
3	+	17	!	31	/	45	=	59	K	73	Y	87	g	101	u				
4	×	18	"	32	0	46	>	60	L	74	Z	88	h	102	v				
5	◇	19	#	33	1	47	?	61	M	75	[	89	i	103	w				
6	⏀	20	$	34	2	48	@	62	N	76	\	90	j	104	x				
7	⧖	21	%	35	3	49	A	63	O	77	]	91	k	105	y				
8	⧗	22	&	36	4	50	B	64	P	78	^	92	l	106	z				
9	⧩	23	’	37	5	51	C	65	Q	79	¯	93	m	107	{				
10	⧈	24	(	38	6	52	D	66	R	80	‘	94	n	108	|				
11	✳	25	)	39	7	53	E	67	S	81	a	95	o	109	}				
12	⧖	26	*	40	8	54	F	68	T	82	b	96	p	110	~				
13	|	27	+	41	9	55	G	69	U	83	c	97	q	111					

Times-Roman: plotnps -EPS -F1 < TABL1.PLT

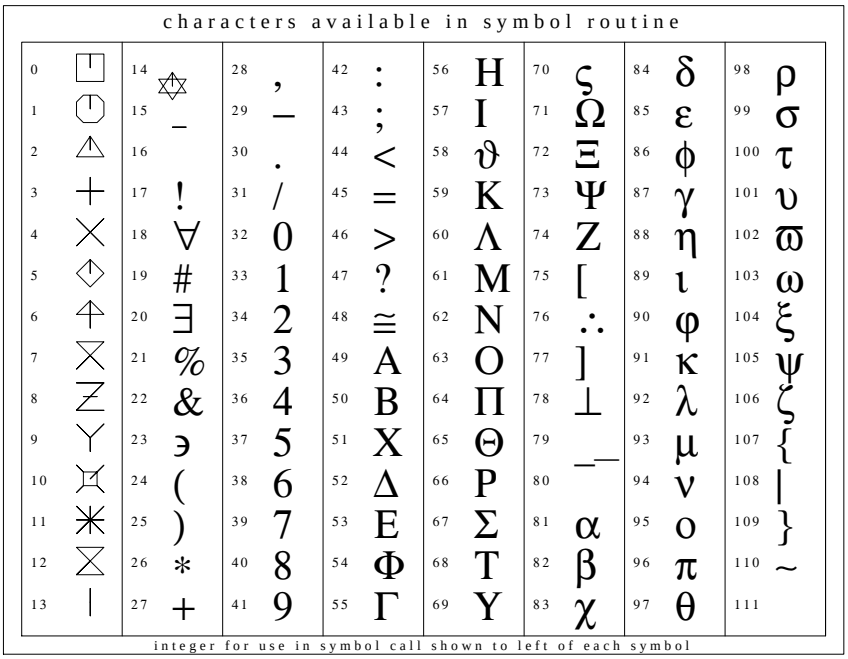


0	□	14	✡	28	,	42	:	56	*H*	70	*V*	84	*d*	98	*r*				
1	○	15	_	29	-	43	;	57	*I*	71	*W*	85	*e*	99	*s*				
2	△	16		30	.	44	<	58	*J*	72	*X*	86	*f*	100	*t*				
3	+	17	*!*	31	*/*	45	=	59	*K*	73	*Y*	87	*g*	101	*u*				
4	×	18	"	32	*0*	46	>	60	*L*	74	*Z*	88	*h*	102	*v*				
5	◇	19	#	33	*1*	47	*?*	61	*M*	75	*[*	89	*i*	103	*w*				
6	⏀	20	$	34	*2*	48	@	62	*N*	76	\	90	*j*	104	*x*				
7	⧖	21	%	35	*3*	49	*A*	63	*O*	77	*]*	91	*k*	105	*y*				
8	⧗	22	&	36	*4*	50	*B*	64	*P*	78	^	92	*l*	106	*z*				
9	⧩	23	’	37	*5*	51	*C*	65	*Q*	79	¯	93	*m*	107	*{*				
10	⧈	24	*(*	38	*6*	52	*D*	66	*R*	80	‘	94	*n*	108	|				
11	✳	25	*)*	39	*7*	53	*E*	67	*S*	81	*a*	95	*o*	109	*}*				
12	⧖	26	*	40	*8*	54	*F*	68	*T*	82	*b*	96	*p*	110	~				
13	|	27	+	41	*9*	55	*G*	69	*U*	83	*c*	97	*q*	111					

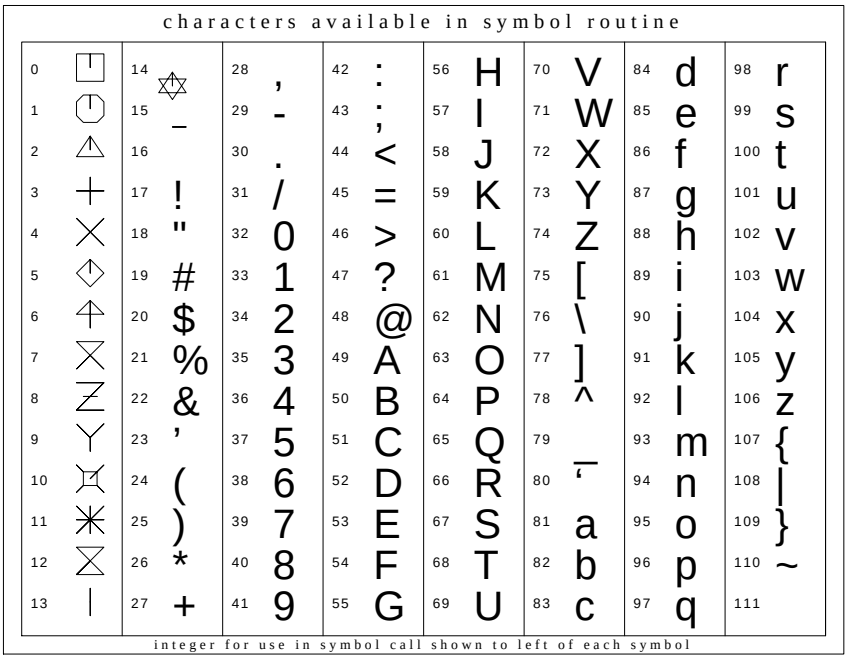
Times-Italic: plotnps -EPS -F2 < TABL1.PLT



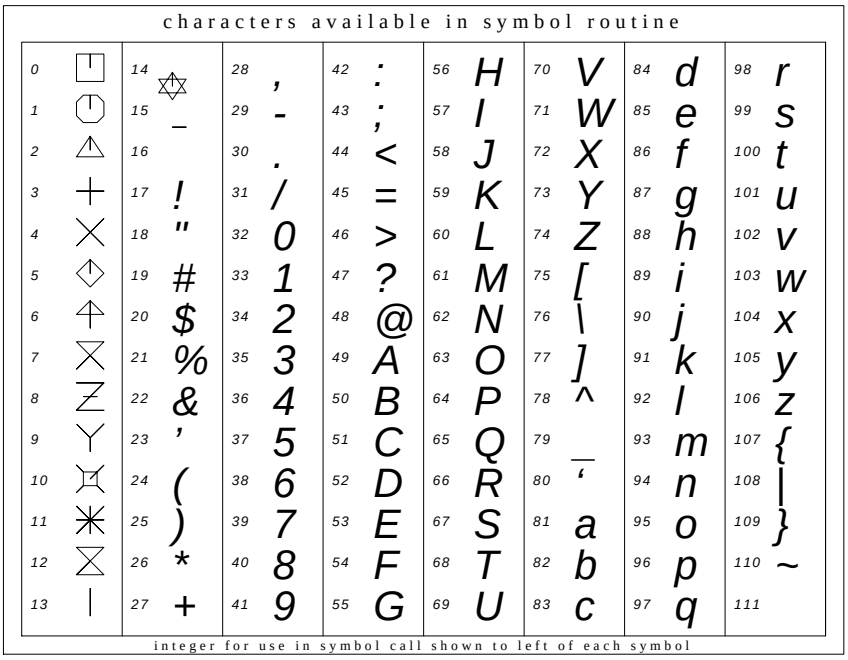
Times-Bold: plotnps -EPS -F3 < TABL1.PLT



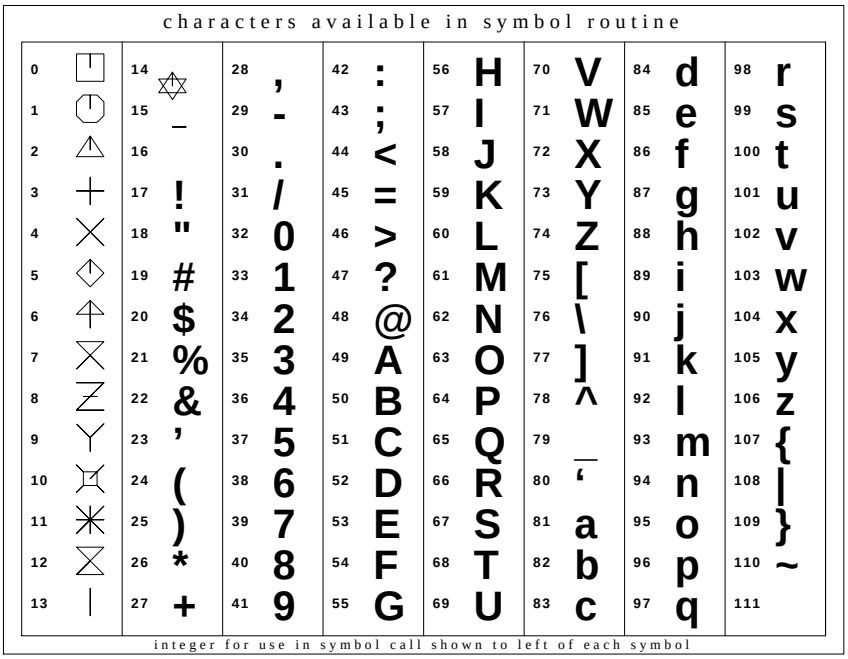
Symbol: plotnps -EPS -F4 < TABL1.PLT



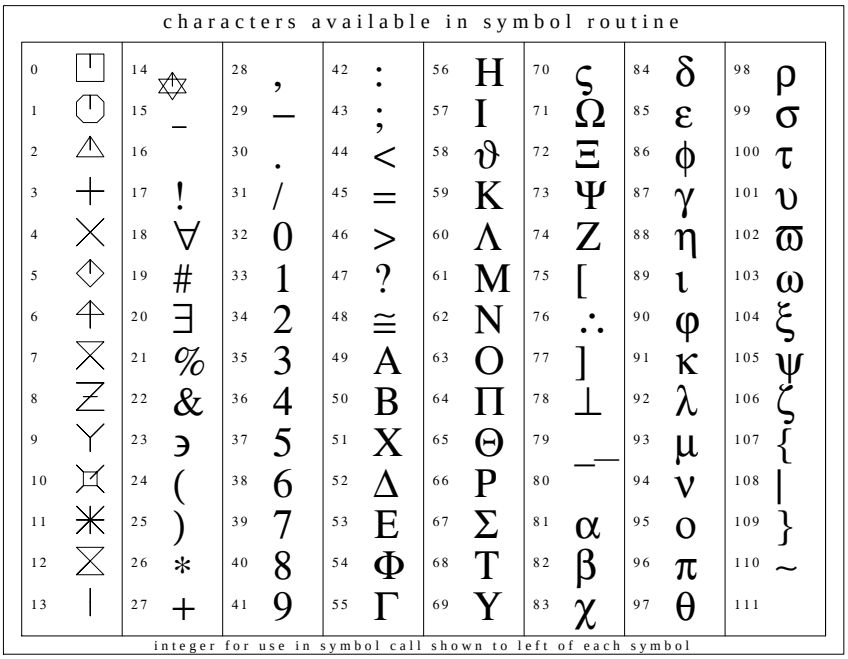
Helvetica: plotnps -EPS -F5 < TABL1.PLT



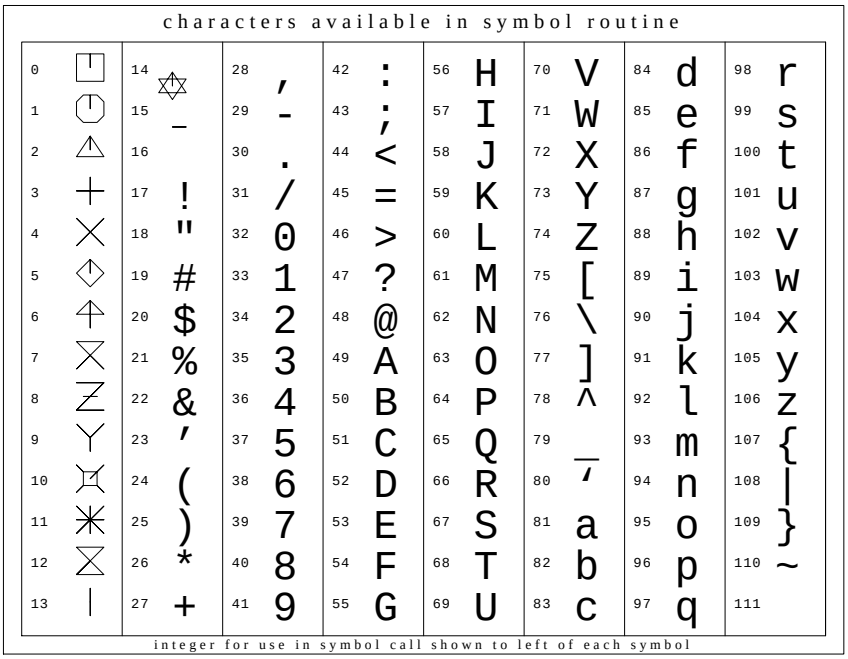
Helvetica-Italic: plotnps -EPS -F6 < TABL1.PLT



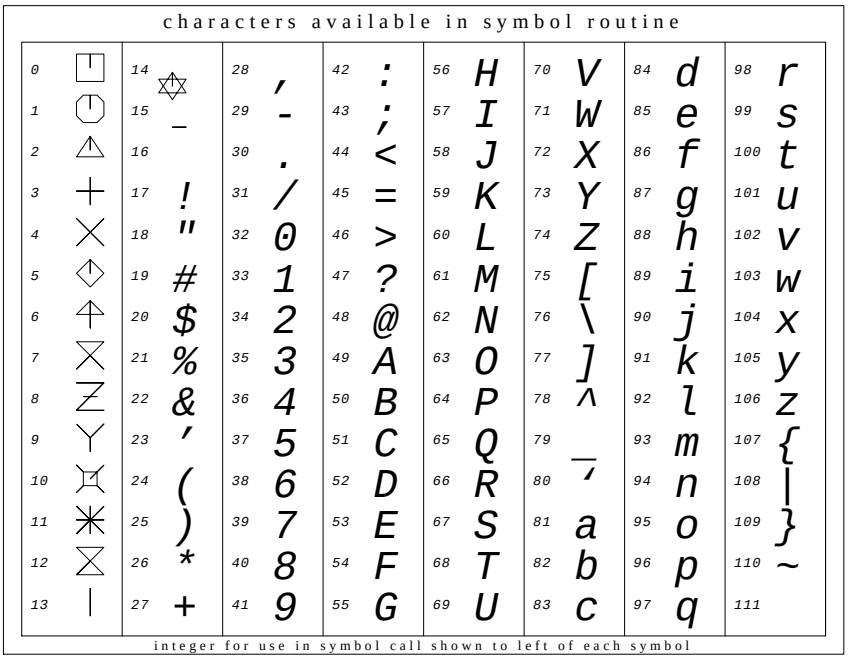
Helvetica-Bold: plotnps -EPS -F7 < TABL1.PLT



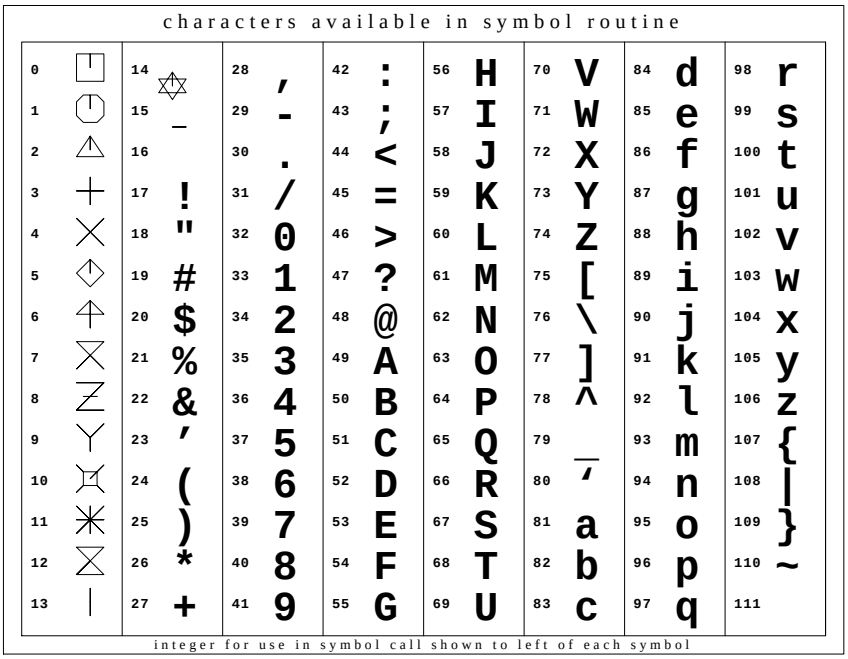
Symbol: plotnps -EPS -F8 < TABL1.PLT



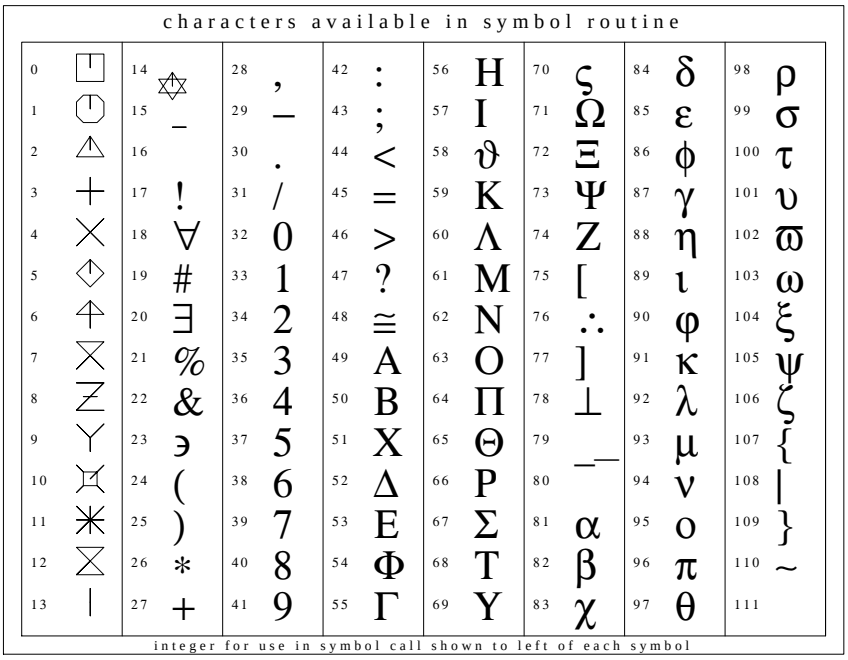
Courier: plotnps -EPS -F9 < TABL1.PLT



Courier-Italic: plotnps -EPS -F10 < TABL1.PLT



Courier-Bold: plotnps -EPS -F11 < TABL1.PLT



Symbol: plotnps -EPS -F12 < TABL1.PLT

CHAPTER 5
EXAMPLES

This chapter provides details on the implementation and use of *CALPLOT* on computers running under UNIX.

1. Compiling

The libraries are compiled for the target machine by editing the Makefile in the source directory. All use of the libraries must be consistent with the compile flags used at this stage. For example, if a *-fswitch* flag is used to make a library, then the same switch must be used when compiling and linking programs to the library. Two sets of libraries are provided: one for a C programming environment and the other for a FORTRAN environment.

C Programs

Two C libraries are provided. **libcalpltc.a** is used to generate the non-interactive device independent plot file. The library, **libcaltekc.a** is used for interactive programs, with output directed to a Tektronix 4014 terminal or emulator. To compile a program *mytest.c*, one would do the following:

```
gcc mytest.c libcalpltc.a -lm -o mytest
```

or

```
gcc mytest.c libcaltekc.a -lm -o mytest
```

If the *CALPLOT* libraries **libcalpltf.a** and **libcaltekf.a** have been installed in the directory **/usr/local/lib**, then one can use

```
gcc mytest.c -lcaltekc -lm -o mytest
```

To execute either program, enter the command

```
mytest
```

Compiling for interactive graphics requires a slightly more complicated command line for X11 and MS Windows. Look at the Makefiles in graphics test directory

`/PROGRAMS.NNN/CALPLOT/testc` for C.

FORTRAN Programs

Two FORTRAN libraries are provided. **libcalplot.a** is used to generate the non-interactive device independent plot file. The library, **libcaltek.a** is used for interactive programs, with output directed to a Tektronix 4014 terminal or emulator. To compile a program *mytest.f*, one would do the following:

```
g77 mytest.f libcalpltf.a -o mytest
```

or

```
g77 mytest.f libcaltekf.a -o mytest
```

If the *CALPLOT* libraries **libcalpltf.a** and **libcaltekf.a** have been installed in the directory **/usr/local/lib**

```
g77 mytest.f -lcaltekf  -o mytest
```

To execute either program, enter the command

```
mytest
```

Compiling for interactive graphics requires a slightly more complicated command line for X11 and MS Windows. Look at the Makefiles in graphics test directory

`/PROGRAMS.NNN/CALPLOT/testf` for FORTRAN.

Many of the test programs are the same for each language to test the

2. Plotting

When the non-interactive programs are run, a device independent plot file is created. According to the argument of the *pinit()* or *pinitf()* library calls, the binary device independent plot file has the name *plotXXXXX*, where *XXXXX* is a unique number, or a user provided name.

To plot one of these files on a device, one executes a command of the form

`plot`*dev [flags] < plotXXXXX*

The *CALPLOT* plot filters that create PostScript, GIF or PNG files or display on Tektronics, X11 or MS Windows are describe din the Appendix.

3. Examples

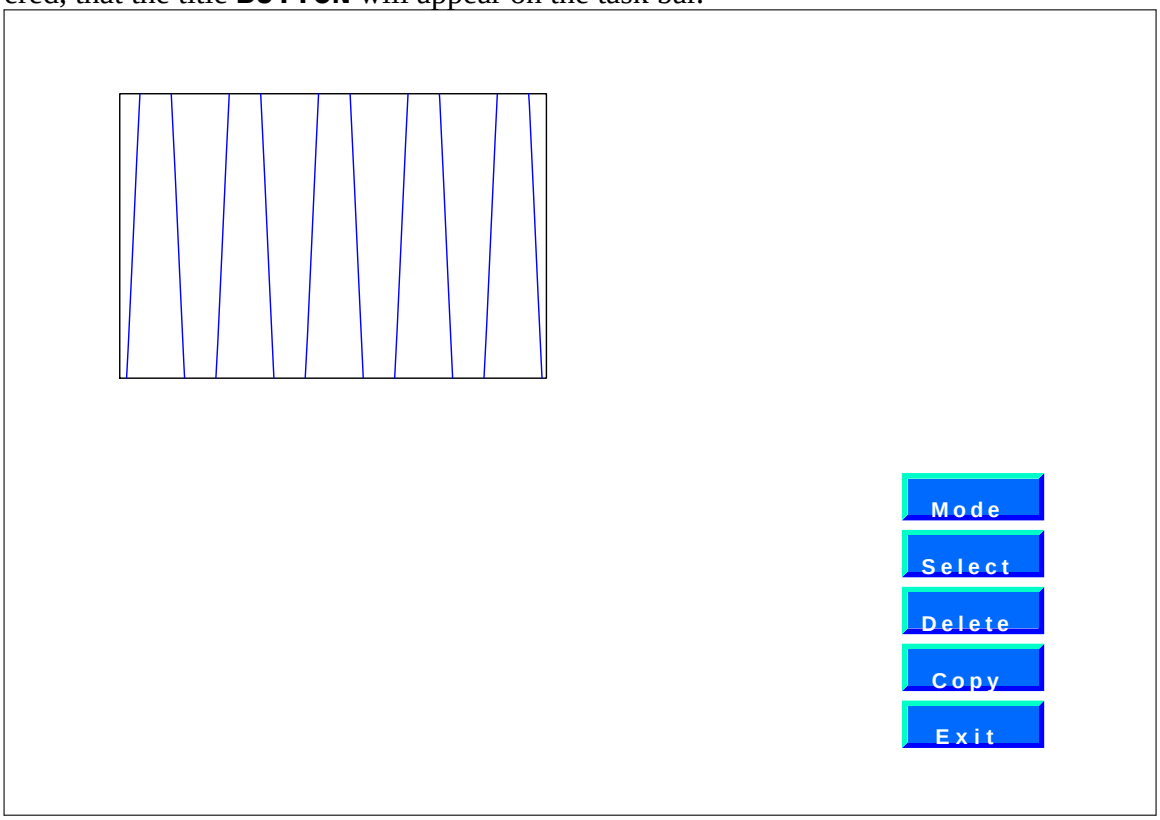
The test programs used to produce these examples are given in the directories

/PROGRAMS.NNN/CALPLOT/testc for the C language, and
/PROGRAMS.NNN/CALPLOT/testf for FORTRAN.

Many of the test programs are the same for each language to test the implmentation of the higher level libraries. Some of the programs were only written in the C language because their implementation is much simpler. These programs test specific commands, and serve as the building blocks for the very useful programs **do_mft** and *do_pom* which are part of the **Computer Programs in Seismology** package.

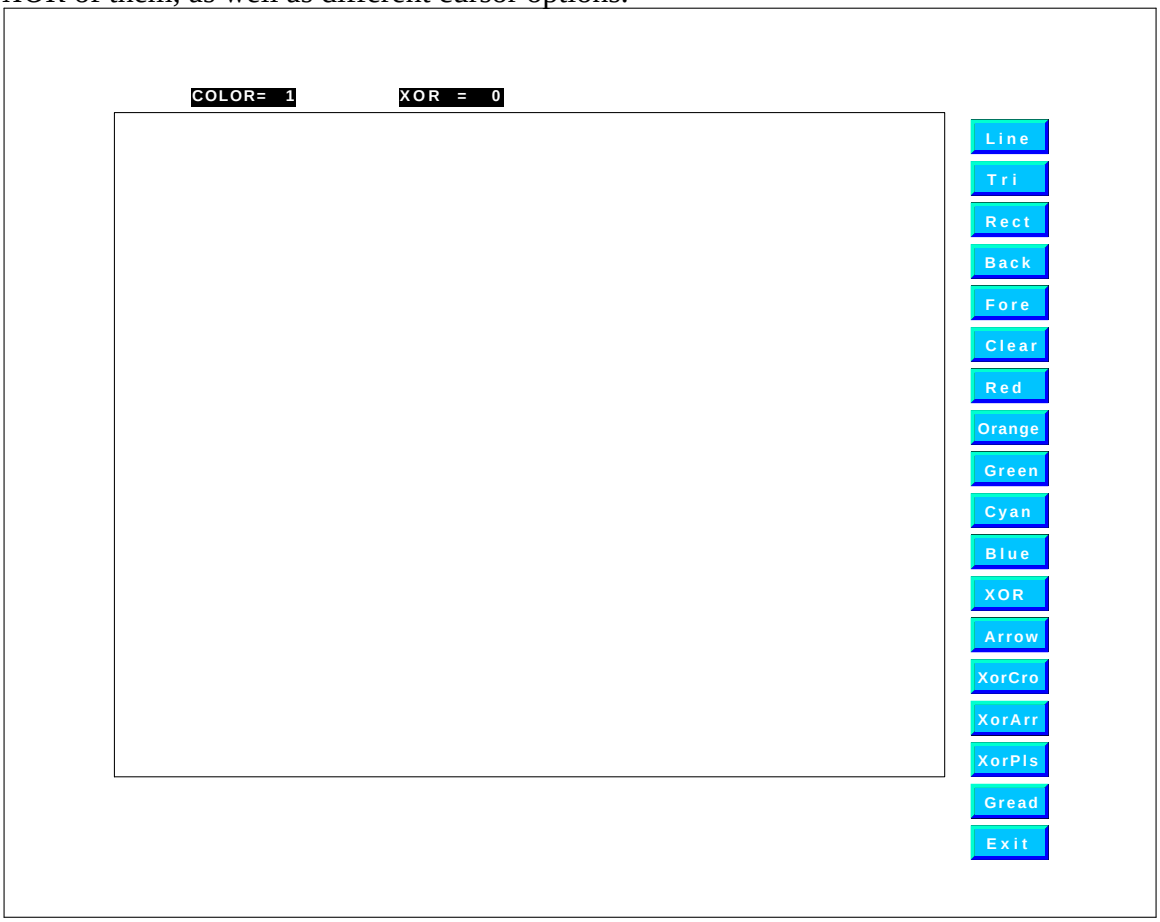
button.c and button.f

This is an interactive program that uses the cursor to select from several buttons. It shows how to constuct buttons (It is easy to change the colors to make the button look as if it is pushed down), and how these buttons can send messages to the top part of the window. The **ginit** initialization is used, which means that if the program window is lowered, that the title **BUTTON** will appear on the task bar.



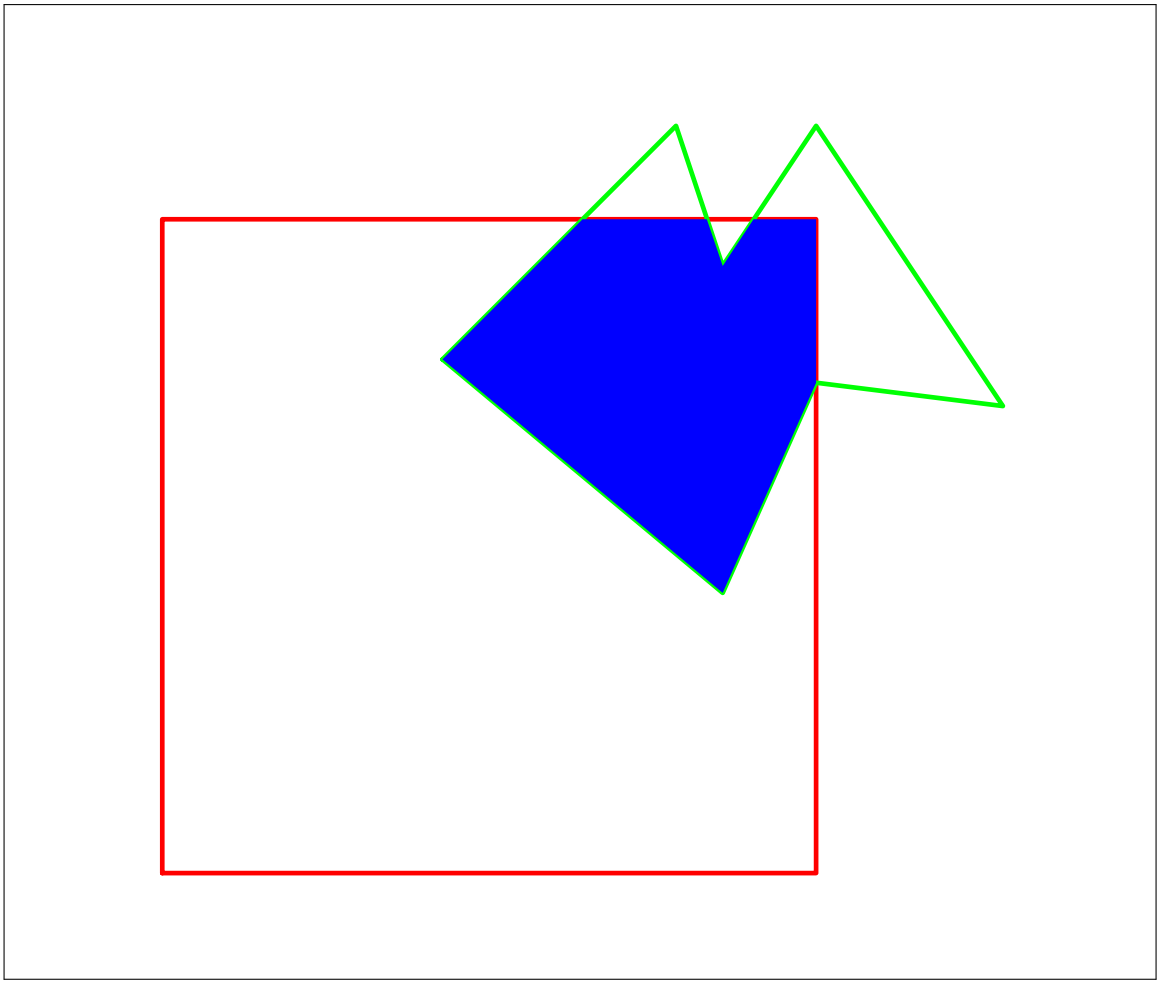
cdraw.c

This is an interactive program that implements buttons and a simple drawing program. The **`gread`** command is tested. The program also tests the various colors and the *XOR* of them, as well as different cursor options.



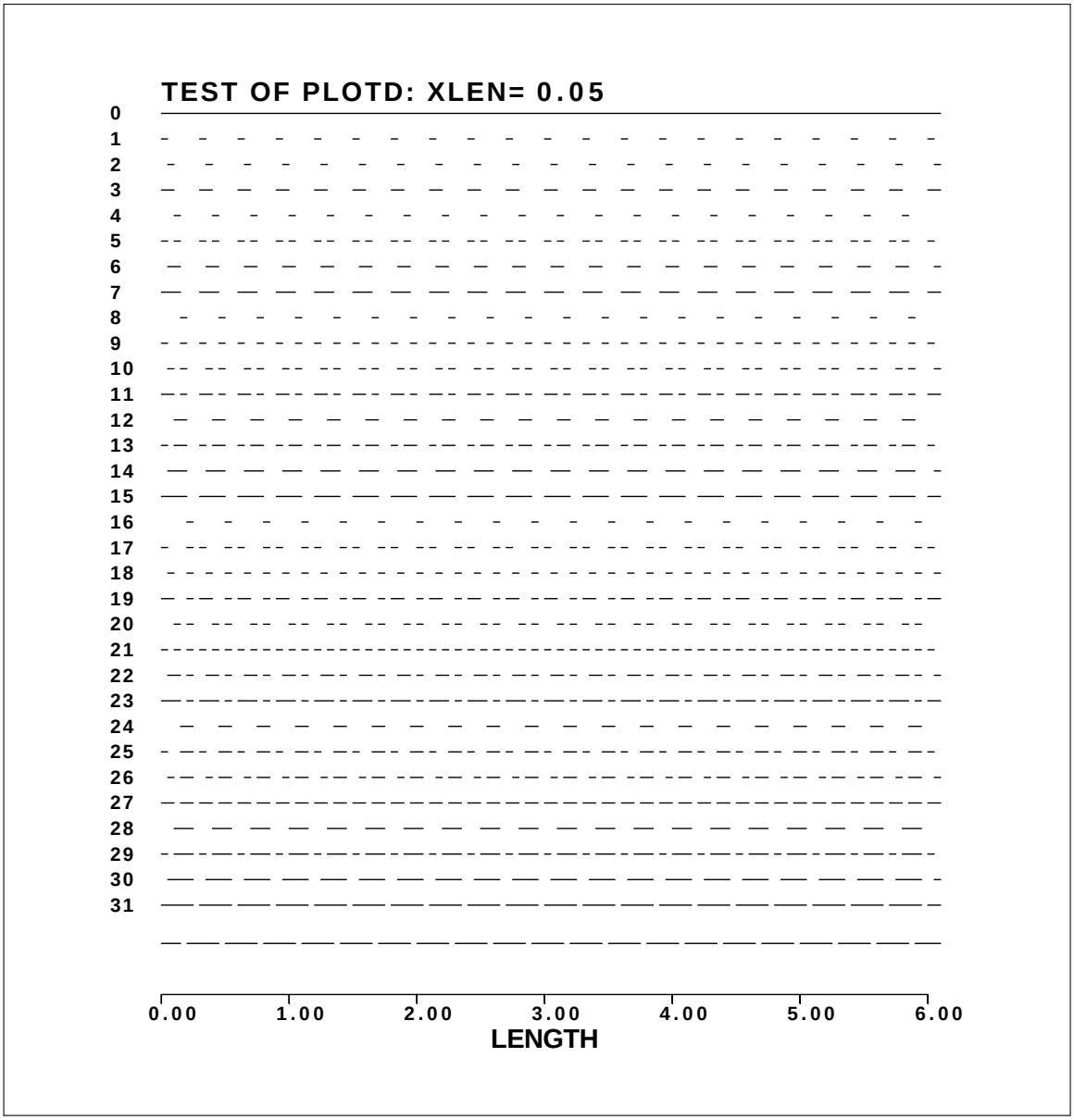
clip.c

This is a simple interactive program that draws a polygon and shades it and shades a square. Clicking the mouse causes the clip region, defined by the **`gclip`** command, to grow. You will initially see the outline of the polygon, which will then be partially shaded as the clip region includes part or all of the polygon. This was written to test low level polygon clipping.



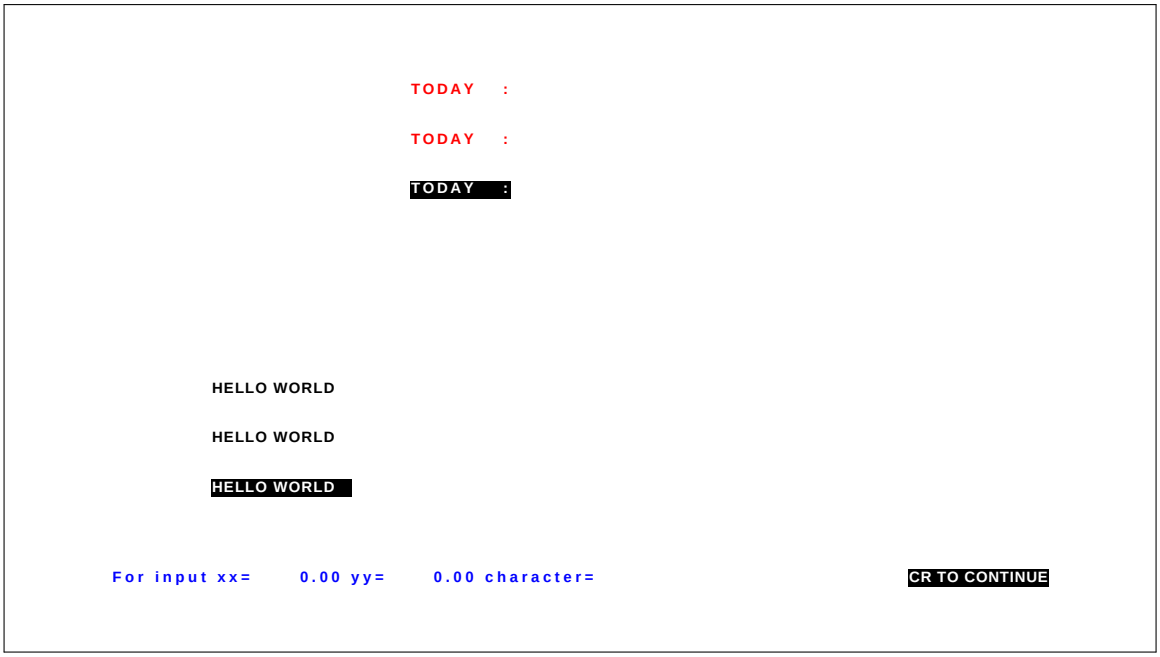
dtest.c or dtest.f

This program tests the **`plotd`** calls which are an extension to the original *CALPLOT* **`plot`** command to permit dashed lines. The usefulness of the example is that it shows the relation between the pattern and the integer defining the pattern.



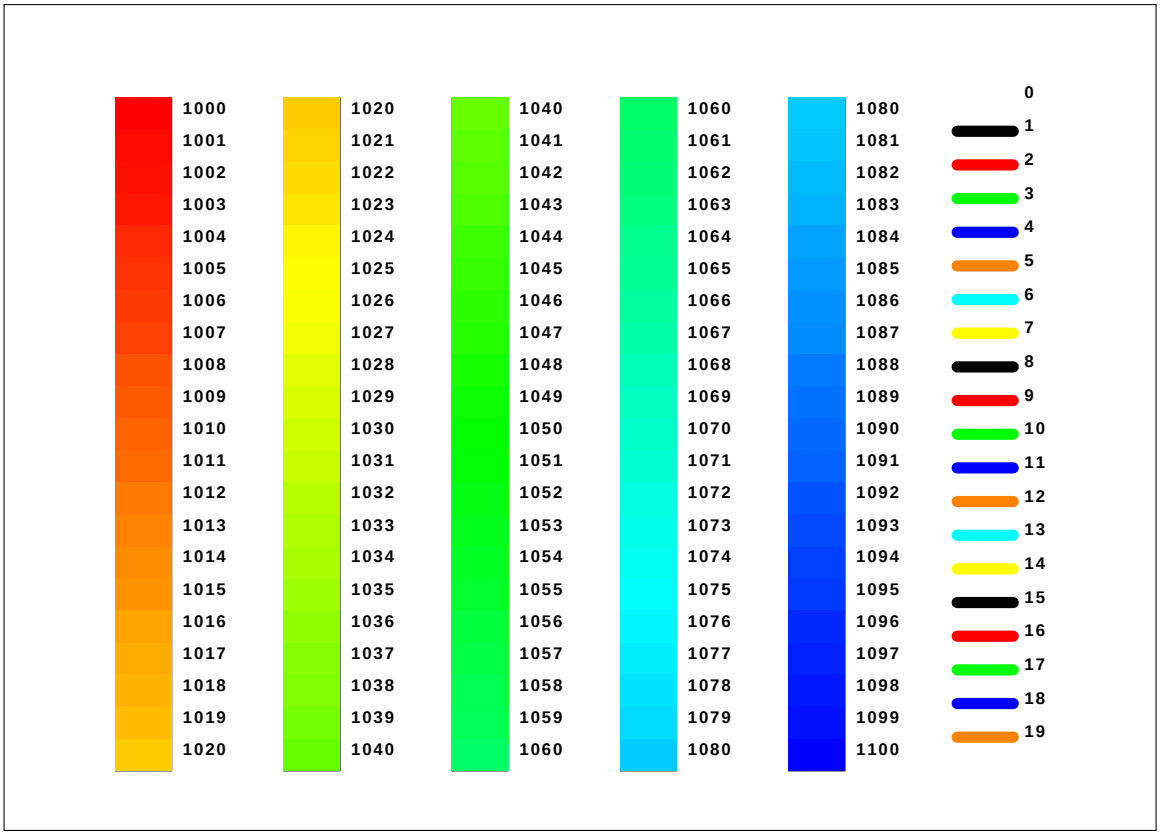
gphtxt.c or gphtxt.f

This interactive program tests the **`gwrtxt`** and **`grdtxt`** mechanisms for text input and output. This uses a hardware font and appears differently in windows of different sizes. Look at **`do_mft`** for a modification that uses a softweare font if the window is too small.



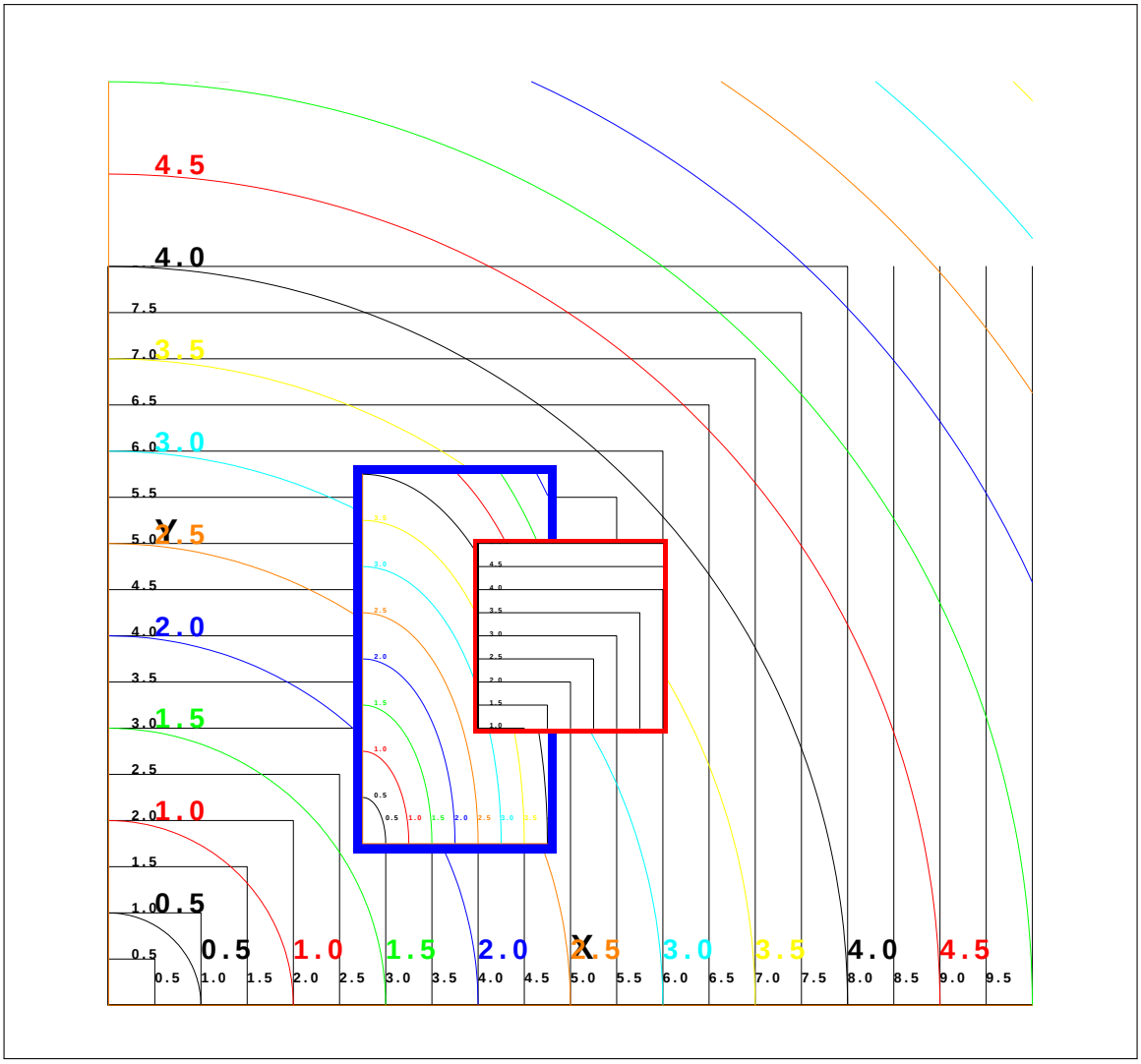
graysc.c or graysc.f

This program shows the color palette used by **`newpen(ipen)`** for *1000* ≤ *ipen* ≤ *1100* and *0* ≤ *ipen* < *1000*. If viewed with different color options for the plot filters, one can discern the meaning of the -K, _G, -KB and -KR flags to **`plotnpsf1`**, **`pltoxvig`** and **`plotmsw`**.



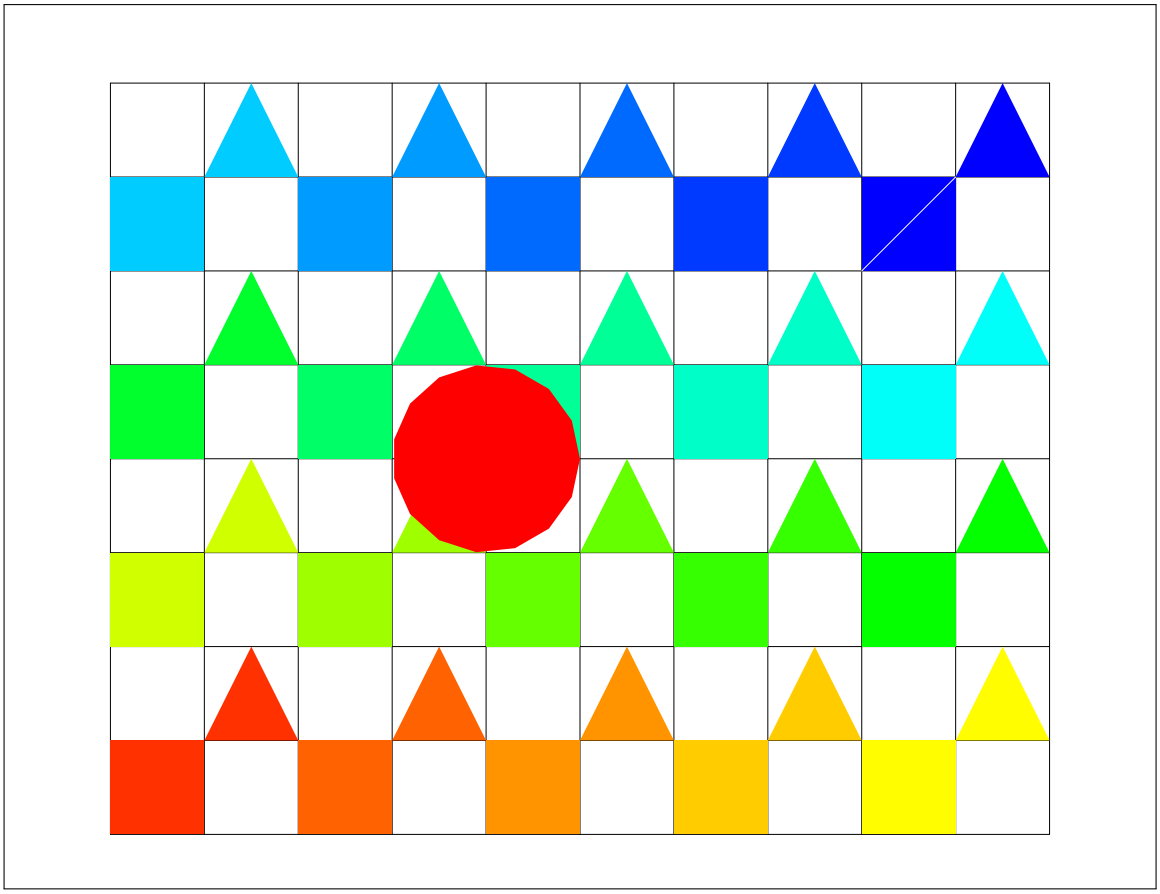
gread.c or gread.f

This tests the ability of the call **`gread()`** to extract portions of a plot from a plot file, to position it on the paper, and to rescale it.



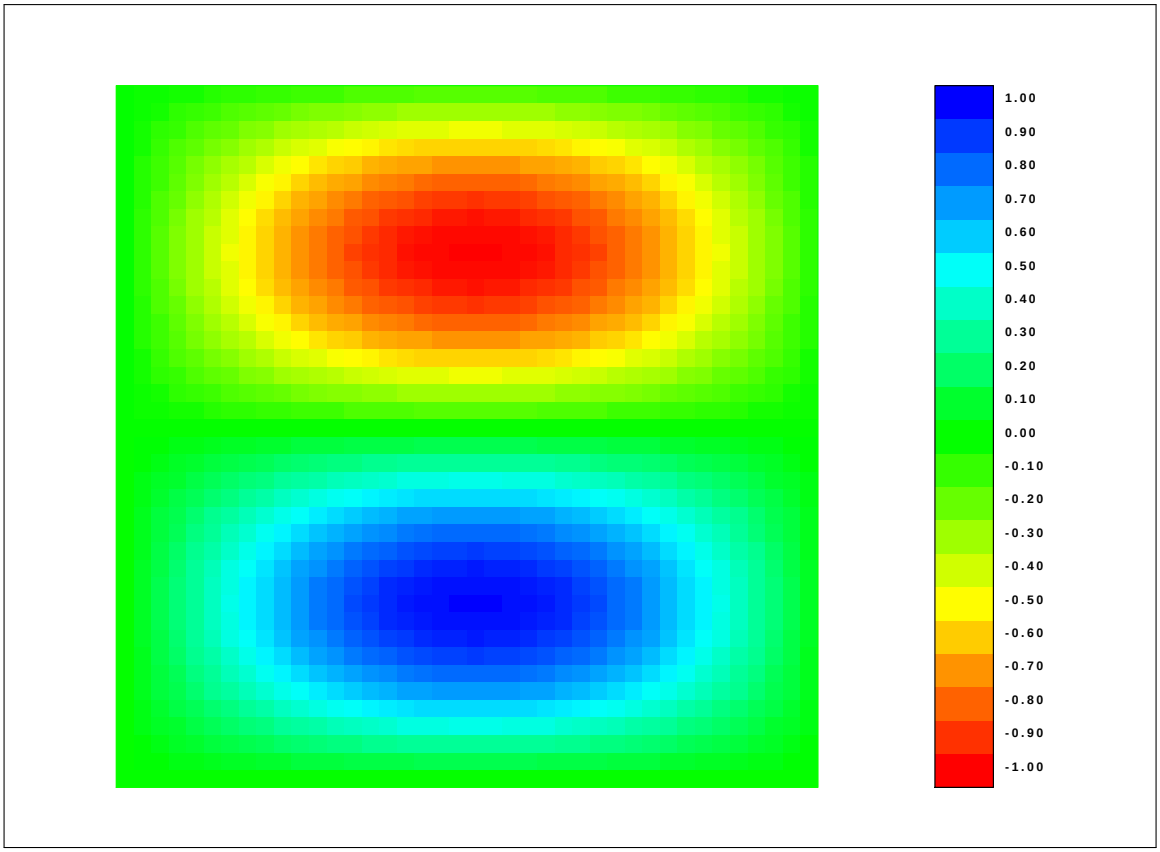
grid.c or grid.f

This program tests the shading routines **`shadep()`**, **`shader()`** and **`shadet()`**.



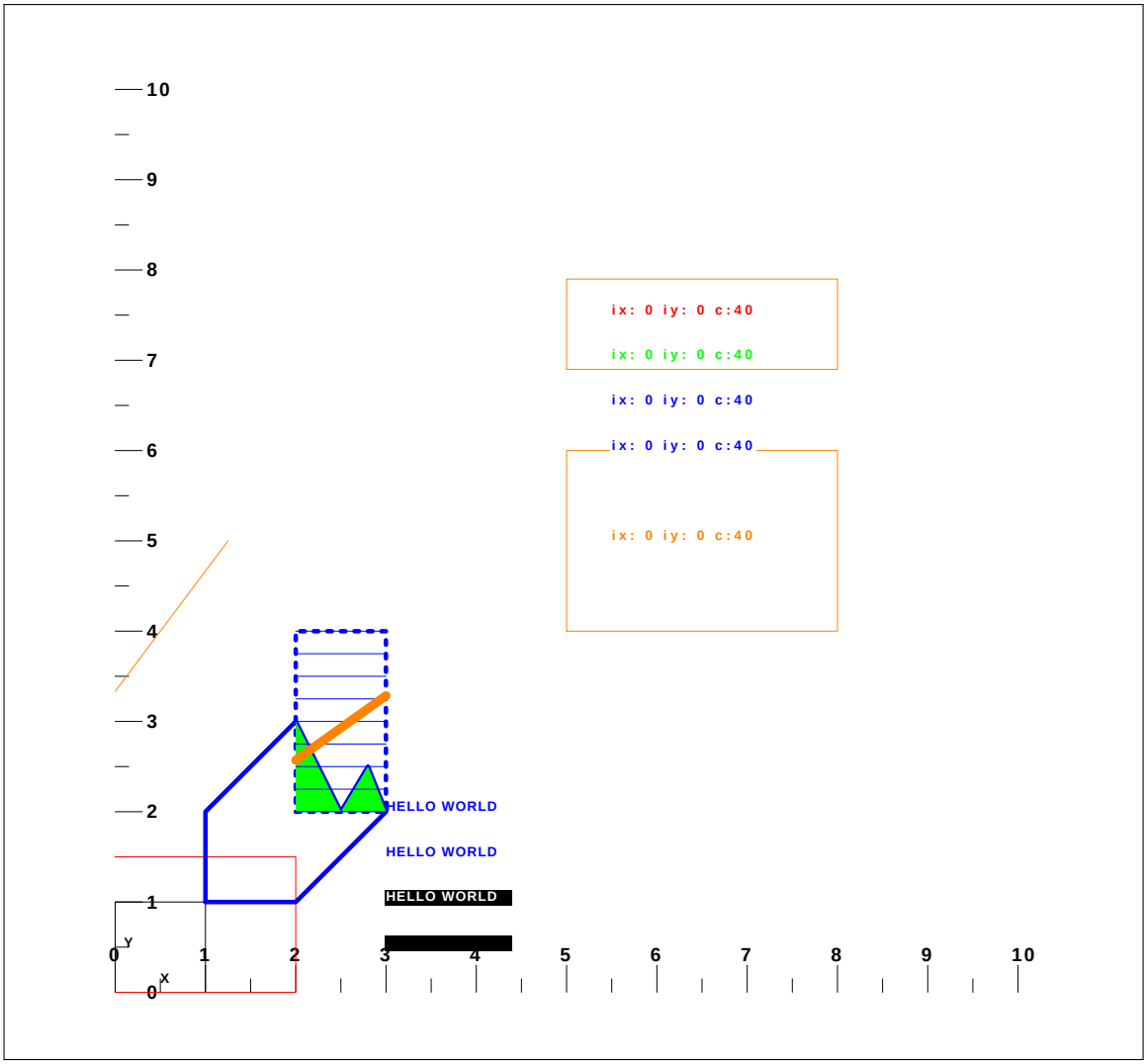
grytst.c or grytst.f

This is an example of how color shading can be used to represent a color surface, by coloring small rectangulat regions. This is *not* an example for a general purpose contouring program.



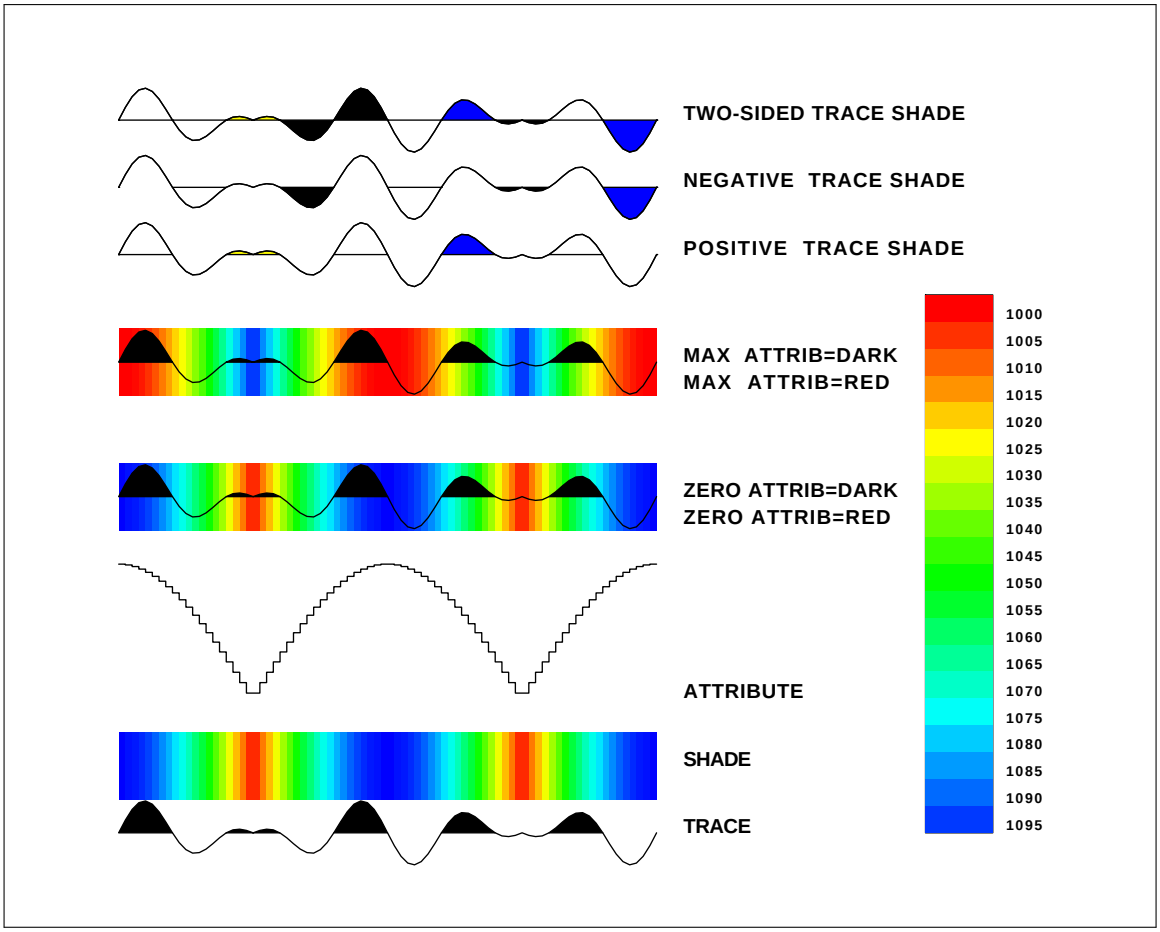
new.c or new.f

This is an important interactive program that tests the different cursors as well as text input. X- and Y-axes are plotted to test the mapping from screen pixel coordinate to the *CALPLOT* coordinate (recall that the 8.0 x 10.0 user screen space is mapped in to 8000 x 10000 *CALPLOT* units which is then mapped into the actual screen size). The initial gursor is a crosshair (see the message bar at top) inside the rectangular clip region and an arrow outside. This will be very useful for seismic waveform analysis. The **`ginfo()`** call is tested as well as all cursor implementations. I believe that the erase character for the **`grdtxt()`** call is a *Backspace*.



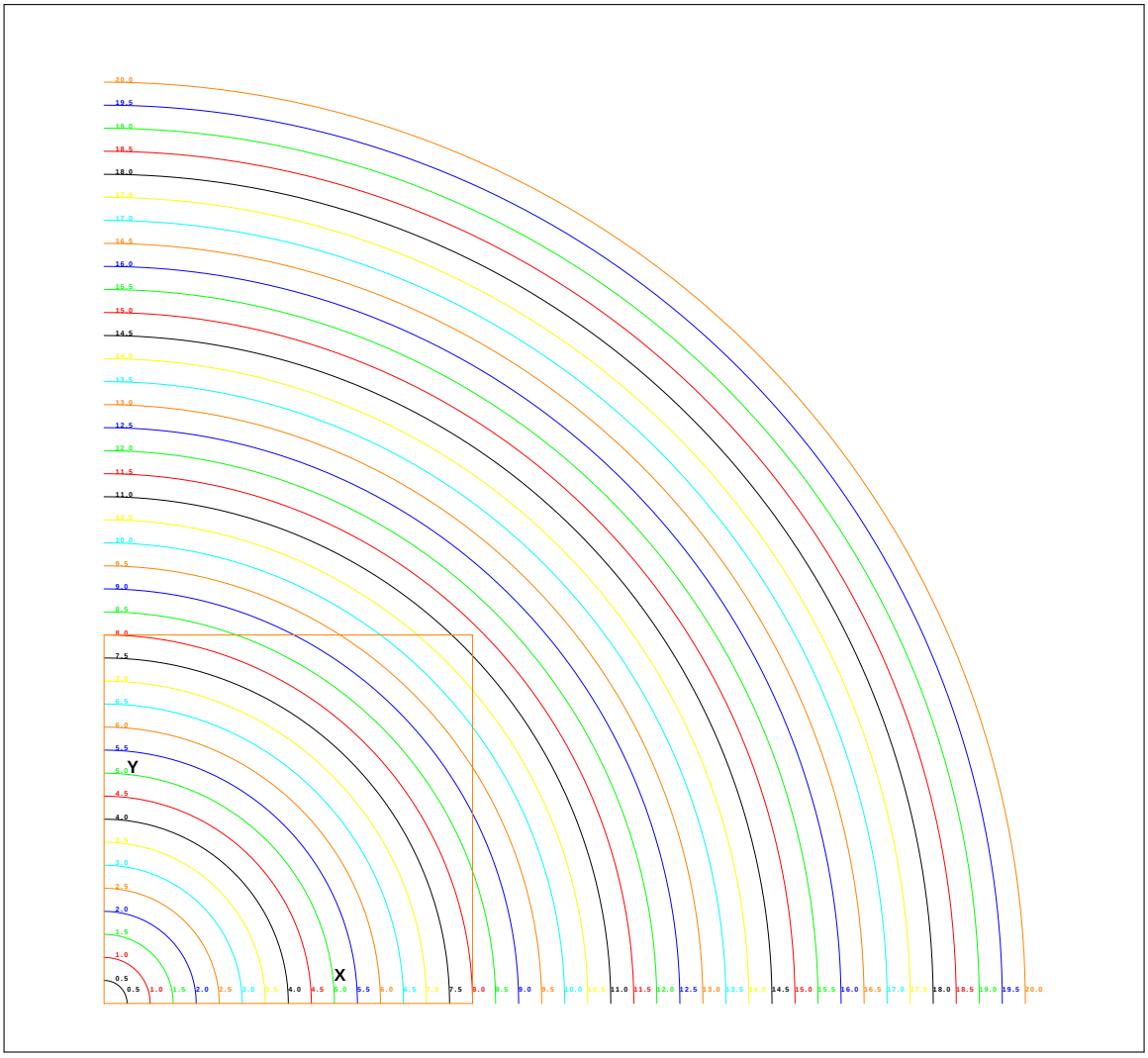
nseitst.c or nseitst.f

This is a test of the seismic trace shading call **`shdsei()`**. The shading is superimposed on colors based on the seismic trace attribute. Code is also provide to shade the trace according to the trace attribute (in exploration seismology this could be the instantaneous frequency or some other feature extraced from the trace). Unfortunately this trace shading works well for a screen display (**`plotxvig`** or **`plotmsw`**) but was a little too hard to implement efficiently for PostScript (**`plotnps`**).



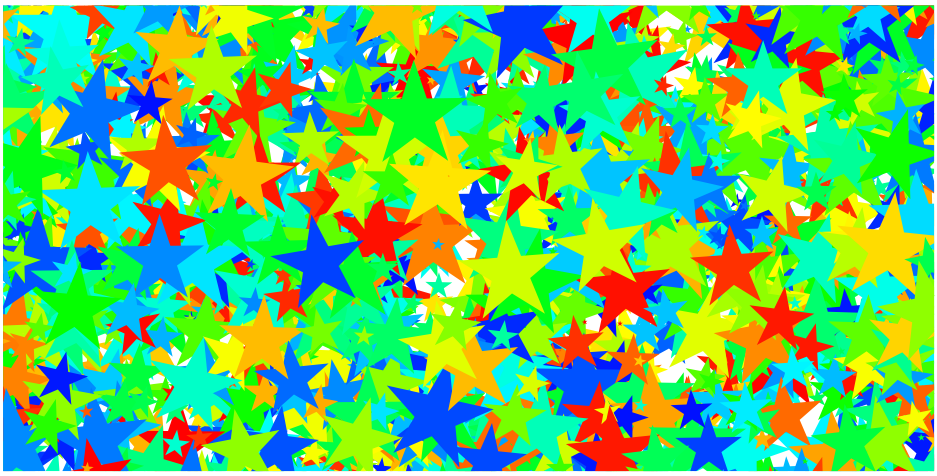
plttst.c or plttst.f

Twenty years ago, it was necessary to write plot filters for each specific dot-matrix printer. This program was written to ensure that if the user called for exactly 1 inch (or cm) in the x- and y-directions, that this would actually be printed, irrespective of the different printer resolutions in the x- and y-directions.



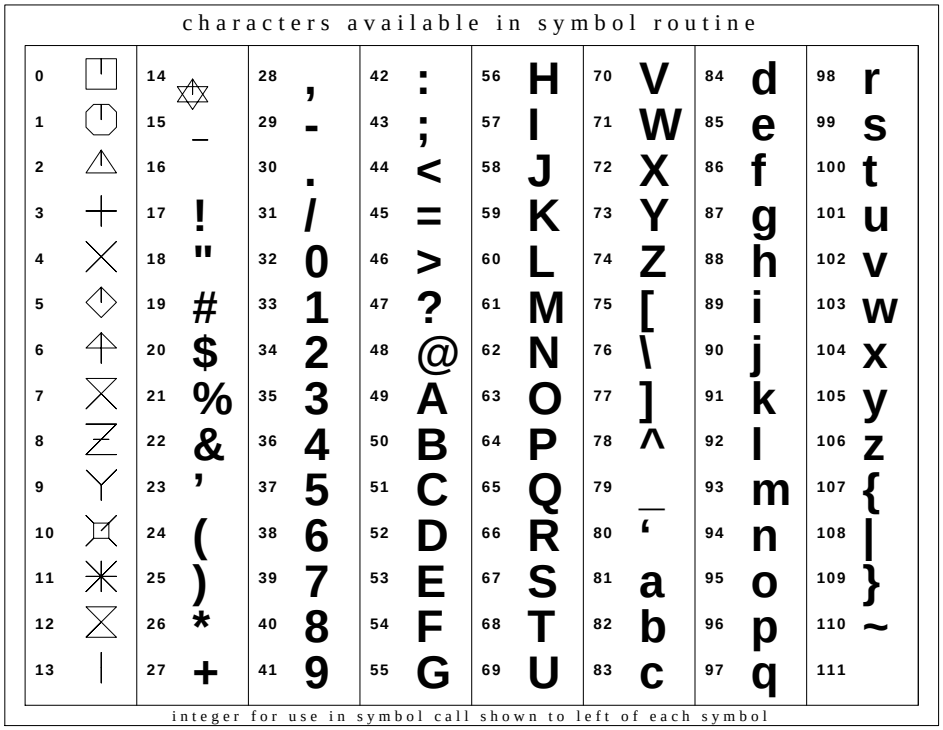
stars.c or stars.f

This draws a large number of randomly colored stars. It serves the rectangular clipping of polygons. When computers were much slower, this was actually a nice display. Perhaps one could introduce a **`sleep`** call to slow things down for an interactive display.



tabl.c or tabl.f

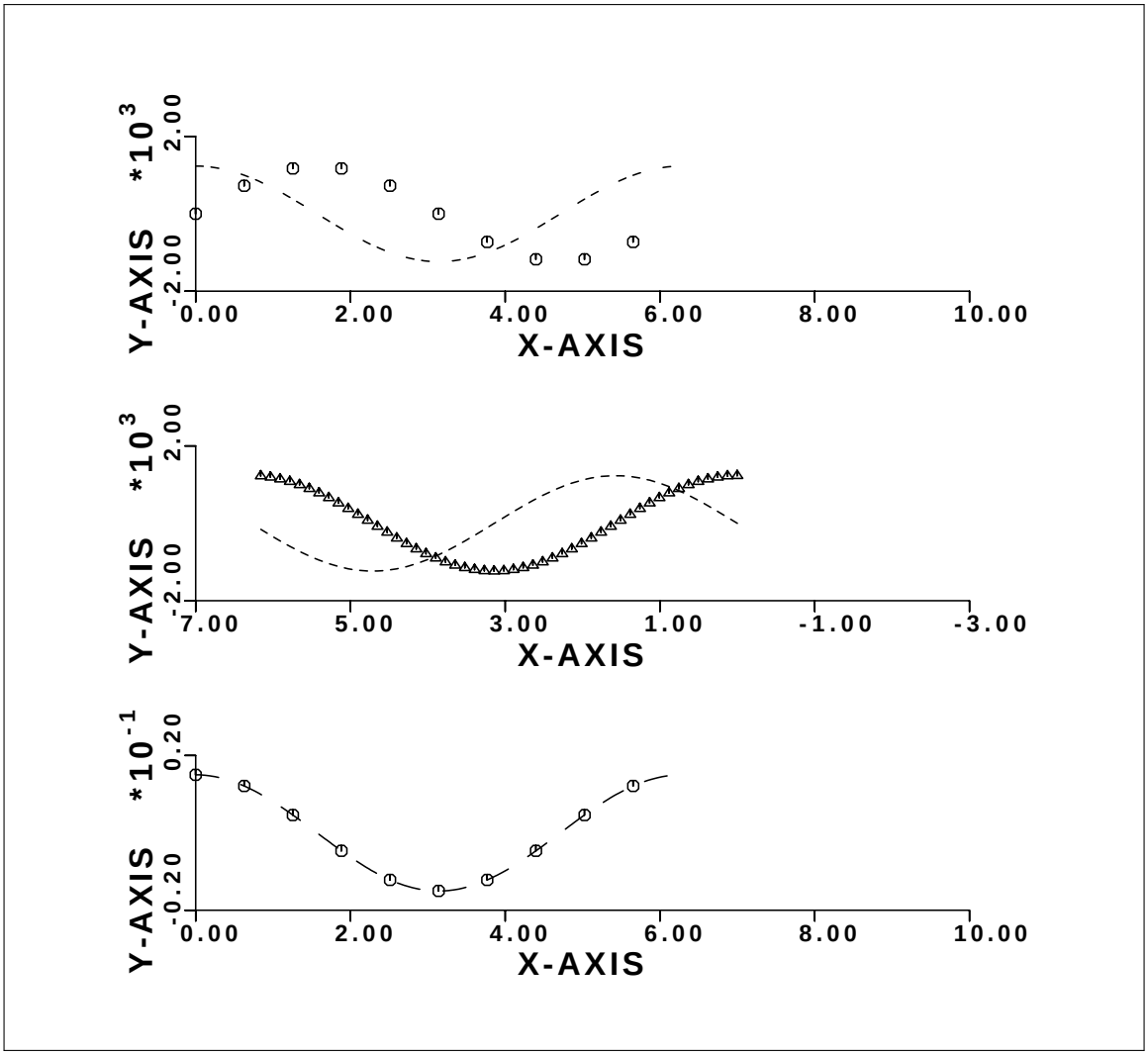
This program shows the character set and symbol set implemented by exercising the **`symbol()`** call. This is identical to the original CALCOMP plotter results. The default character can be overridden by the **`-F`** flag of the plot filter. The **`number()`** call is also tested as well as the *CALPLOT* extersions to permit computer notation of the form *±1.1234E±05* and scientific notation of the form $+-1.1234x10^{\pm 05}$.



0		14		28	,	42	:	56	H	70	V	84	d	98	r
1		15	_	29	-	43	;	57	I	71	W	85	e	99	s
2		16		30	.	44	<	58	J	72	X	86	f	100	t
3		17	!	31	/	45	=	59	K	73	Y	87	g	101	u
4		18	"	32	0	46	>	60	L	74	Z	88	h	102	v
5		19	#	33	1	47	?	61	M	75	[	89	i	103	w
6		20	$	34	2	48	@	62	N	76	\	90	j	104	x
7		21	%	35	3	49	A	63	O	77	]	91	k	105	y
8		22	&	36	4	50	B	64	P	78	^	92	l	106	z
9		23	’	37	5	51	C	65	Q	79	_	93	m	107	{
10		24	(	38	6	52	D	66	R	80	‘	94	n	108	|
11		25	)	39	7	53	E	67	S	81	a	95	o	109	}
12		26	*	40	8	54	F	68	T	82	b	96	p	110	~
13		27	+	41	9	55	G	69	U	83	c	97	q	111	

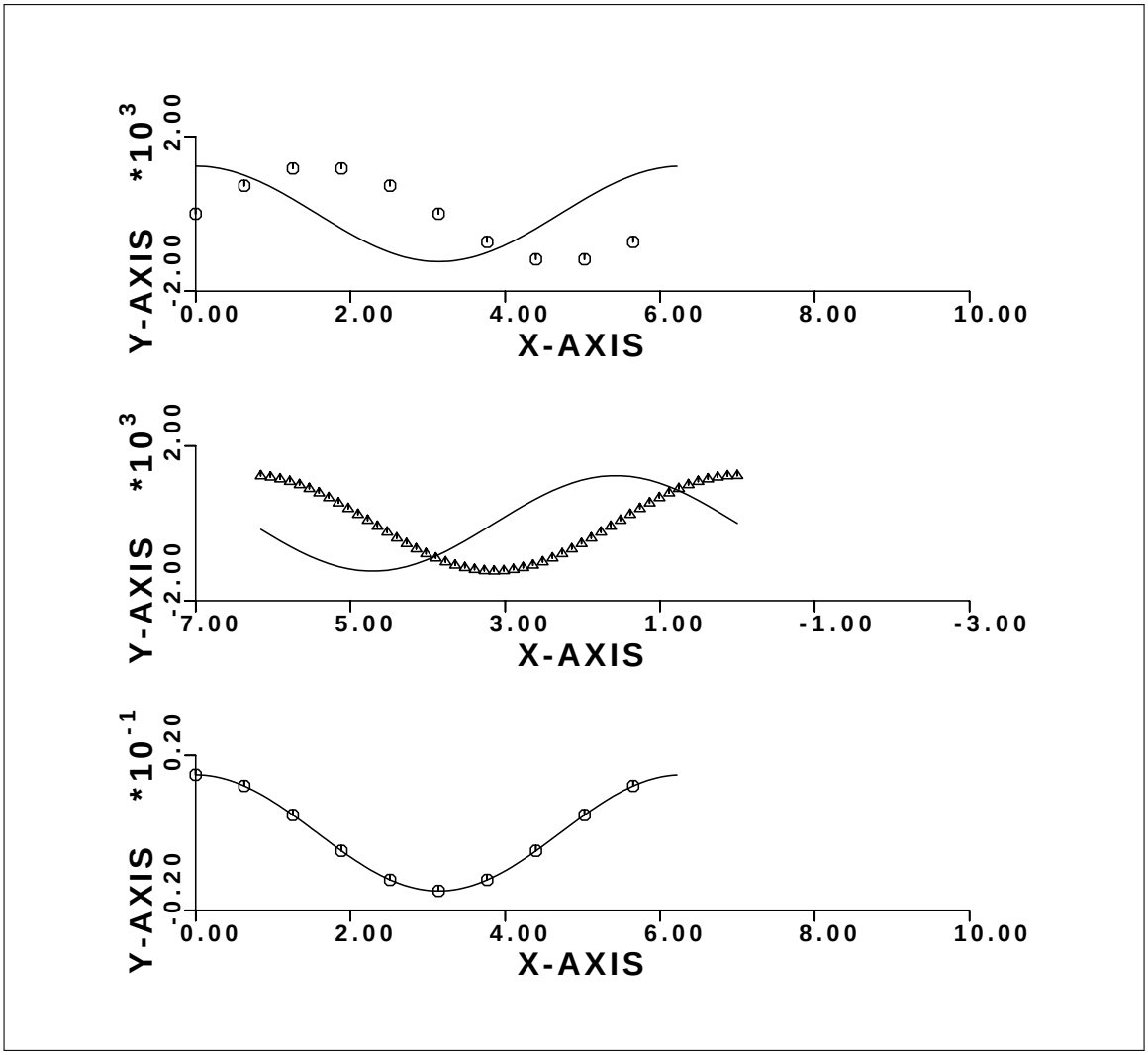
testld.c or testld.f

This tests the extension of the CALCOMP **`line()`** call to permit dased lines.



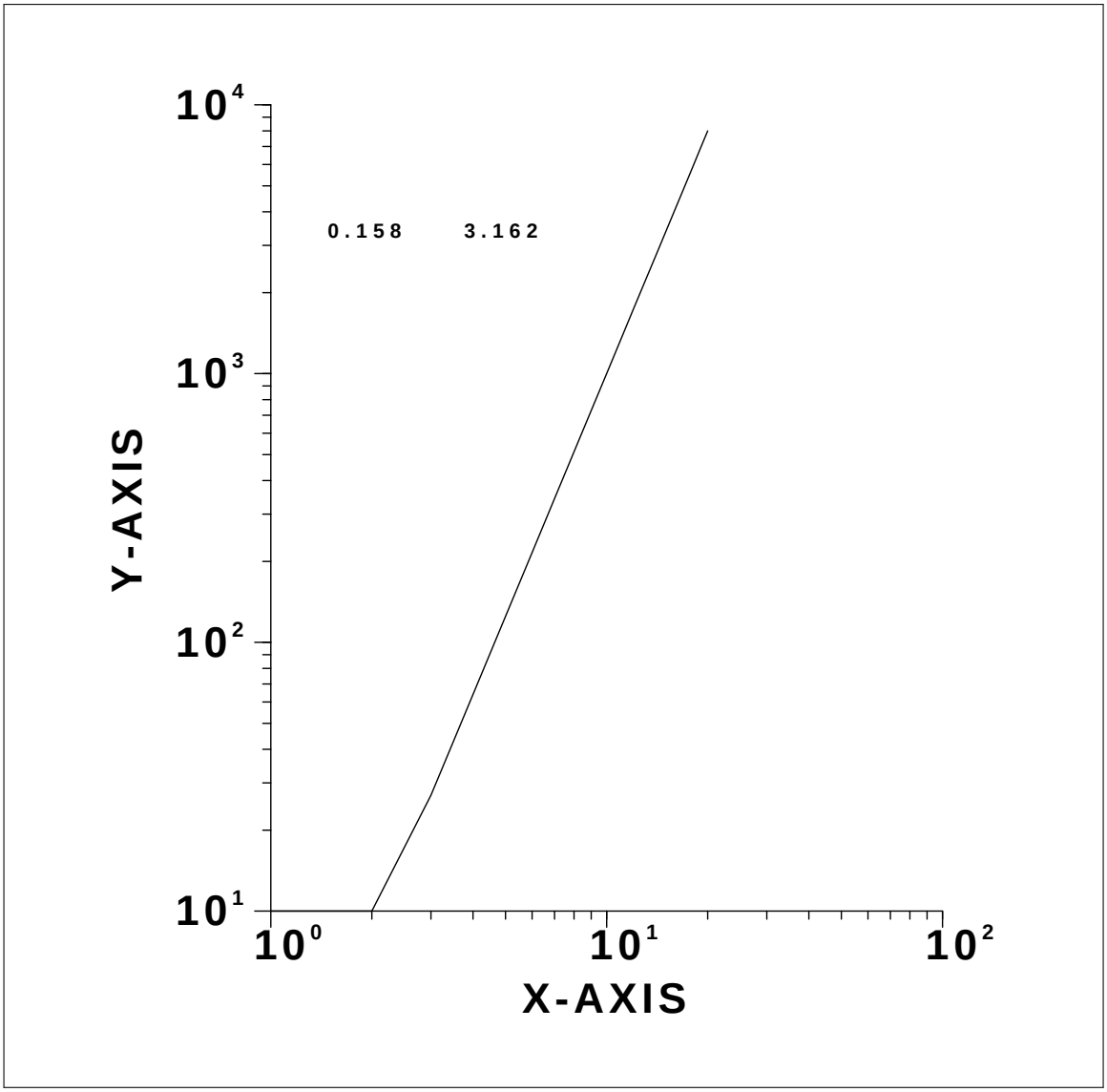
testl.c or testl.f

This tests the implementation of the CALCOMP **`line()`** call.



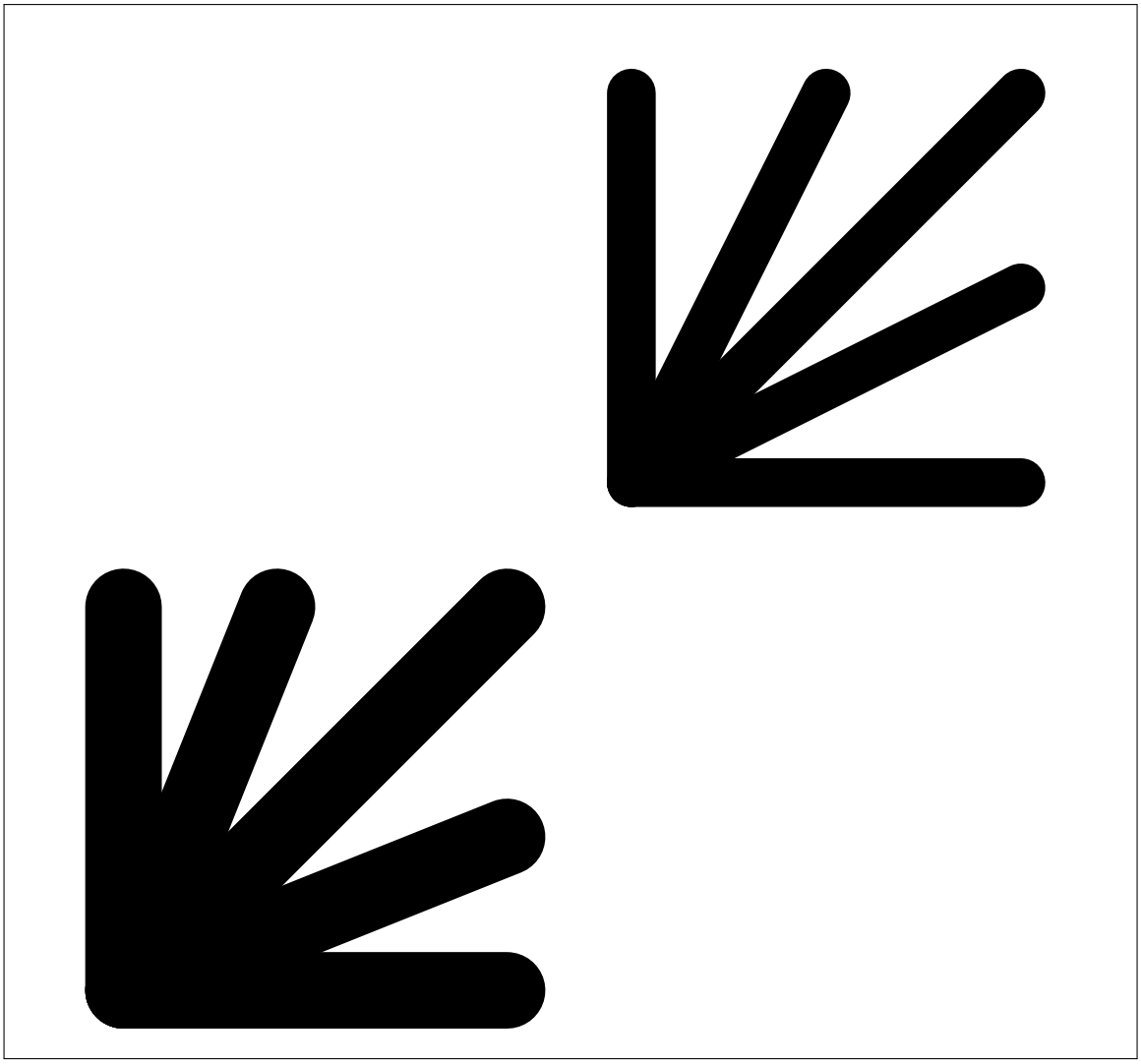
tstcur.c or tstcur.f

This interactive program plots the function $y = x^3$ for various combinations of linear or logarithmic axes. The routines **`pltscl()`**, **`algaxe()`**, **`pltlog()`** and **`pltlgd()`** are used to make the figures. The mouse cursor is used to select a point on the curve of the cubic function and the *user* coordinate are returned. This is a test of the routine **`curuxy`** to correctly return these values. If you select the end fo the curve, the values *(20,8000)* should be written to the screen. A second mouse click advances to the next figure.



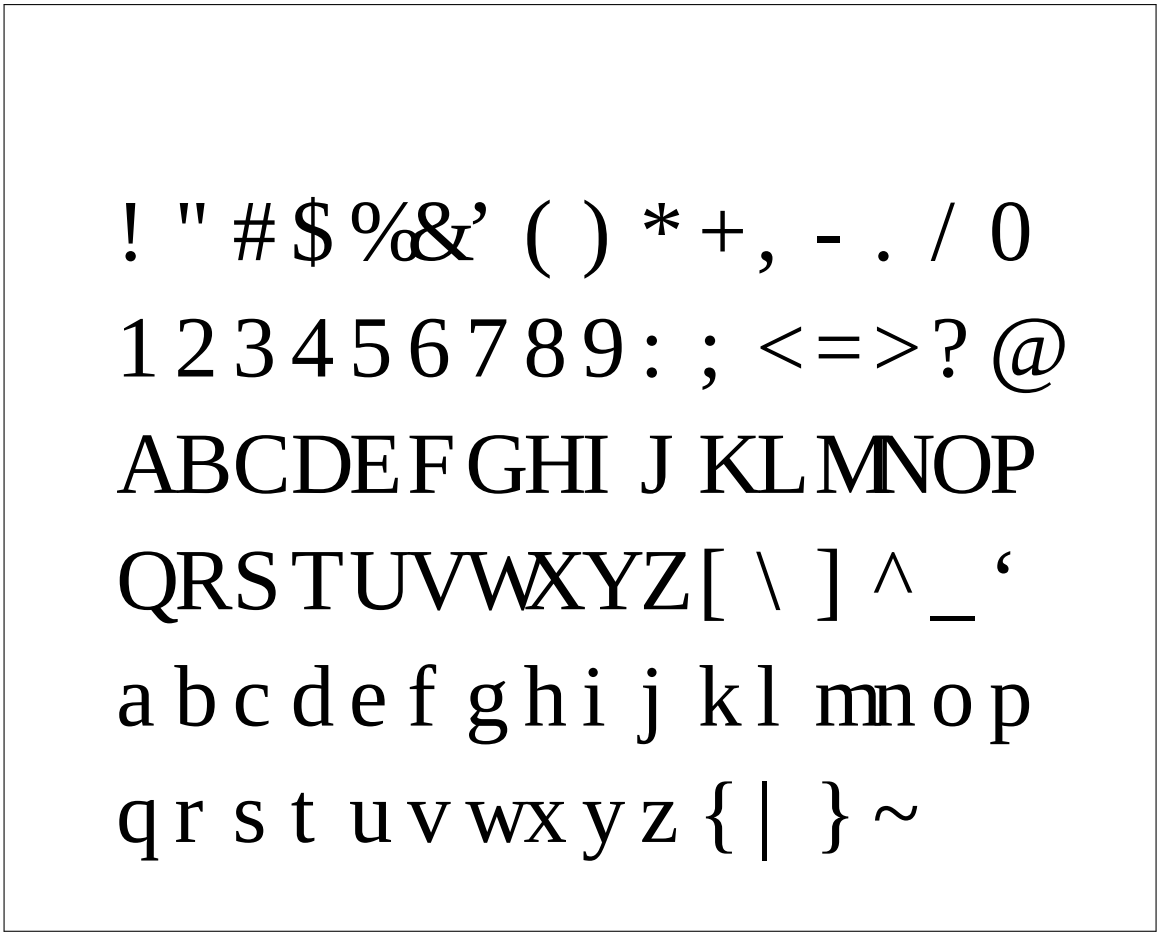
wid.c or wid.f

This unexciting figure draws lines with different widths at different angles. Its original purpose was to ensure that the line width was the same irrespective of the angle at which the line was drawn and the printer resolution.



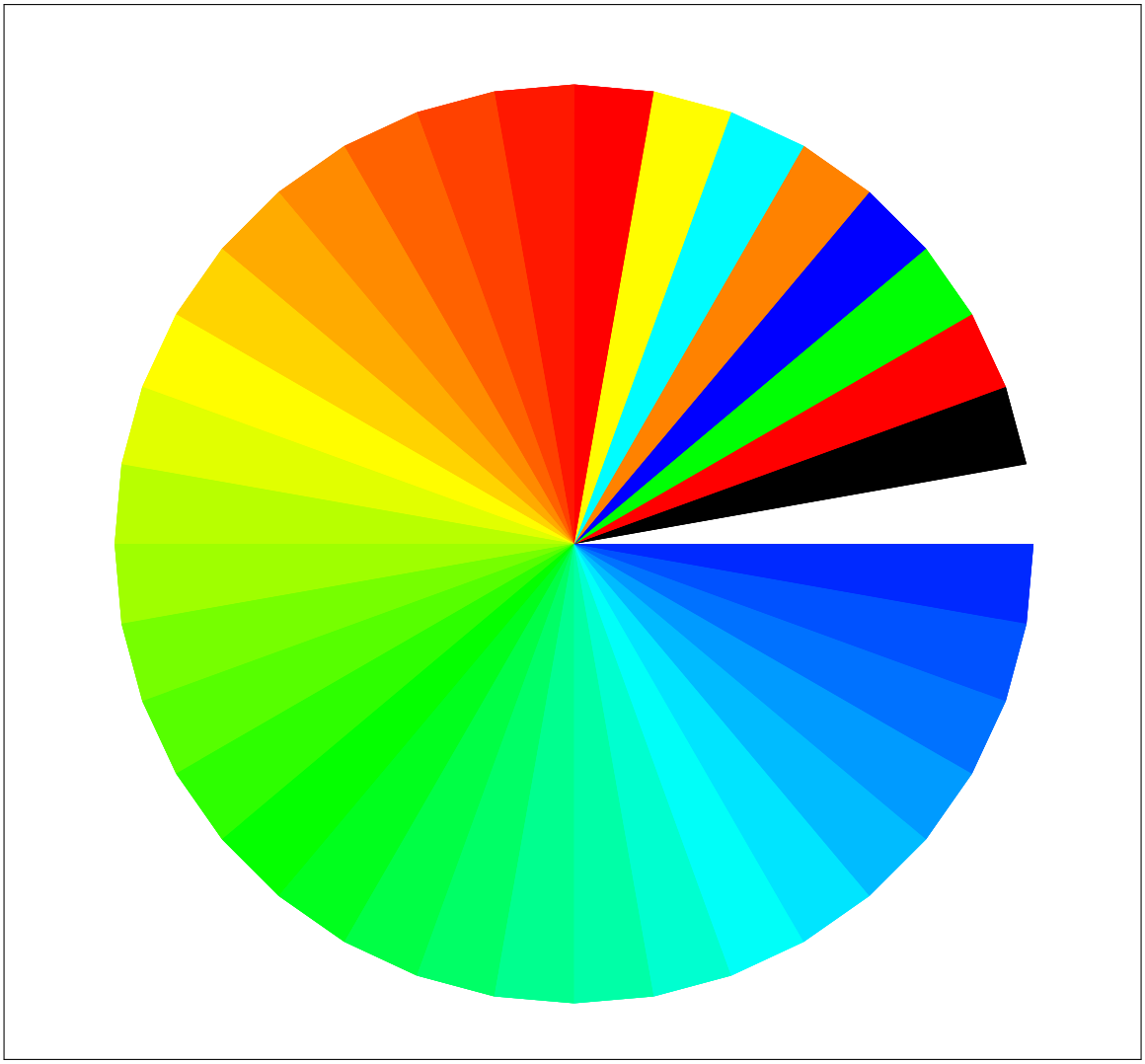
width.c or width.f

This tests the **`gwidth()`** effect on fonts, both for the vector fonts used for the display and PostScript.



xor.c

This draws a sequence of colored sectors on the screen to test the XOR properties of the graphics. On older slower computers you would see a pinwheel effect as the colors were erased. This also requires some time a delay mechanism.



CHAPTER 6
GRAPHSUB ROUTINES

Rather than make the library more complicated that its original implementation and its necessary extensions, additional routines were developed to support the graphics of *Computer Programs in Seismology*. In the distributions, these are in **PROGRAMS.xxx/SUBS** as **grphsubf.f** in FORTRAN and **grphsubc.c** with the include file **graphsub.h** in C. These additional routines were created to simplify programming and to support different axis styles.

1. C Routines

gbox

Draw a box

```
void gbox(float xl,float yl,float xh,float yh);
```

xl	*x-coordinate of the lower left corner*
yl	*y-coordinate of the lower left corner*
xh	*x-coordinate of the upper right corner*
yh	*y-coordinate of the upper right corner*

gcent

Draw text centered at the coordinates *(xx, yy)*.

```
void gcent(float xx,float yy,float ht,char *string,float angle);
```

xx	*x-coordinate of the lower left corner of the first character*
yy	*y-coordinate of the lower left corner of the first character*
ht	*height if the character*
string	*character string to tbe plotted*
angle	*angle at which the string is plotted measured from x-axis*

gright

Draw text right justified at the coordinates *(xx, yy)*.

```
void gright(float xx,float yy,float ht,char *string,float angle);
```

xx	*x-coordinate of the lower left corner of the first character*
yy	*y-coordinate of the lower left corner of the first character*
ht	*height if the character*
string	*character string to tbe plotted*
angle	*angle at which the string is plotted measured from x-axis*

gleft

Draw text left justfied at the coordinates *(xx, yy)*

```
void gleft(float xx,float yy,float ht,char *string,float angle);
```

xx	*x-coordinate of the lower left corner of the first character*
yy	*y-coordinate of the lower left corner of the first character*
ht	*height if the character*
string	*character string to tbe plotted*
angle	*angle at which the string is plotted measured from x-axis*

dology

Plot a logarithmic y-axis. This differs from the **algaxe** routine in that the starting and end values need not be exact powers of 10.

```
void dology(float x0,float y0,float yleng,float ymax,float ymin,
            float sizey,int ticlft,int lablft,int dopow,
            int ly, char *str);
```

x0	*Coordinates of bottom of axis*
y0	
yleng	*Length of y-axis*
ymax	*Maximum value of y-axis, must be > 0*
ymin	*Minimum value of y-axis, must be > 0*
sizey	*Size of numbers*
ticlft	*If > 0, number tics are to the left in y-direction*
lablft	*If > 0, place numbers to left of Y-axis*

dopow	*If > 0, annotate with numbers*
ly	*Number of characters in title*
titley	*Title string*

---

dologx

Plot a logarithmic x-axis. This differs from the **algaxe** routine in that the starting and end values need not be exact powers of 10.

```
void dologx(float x0,float y0,float xleng,float sxmax,float sxmin,
            float sizex,int ticup,int labtop,int dopow,
            int lx, char *str);
```

x0	*Coordinates of left side of axis*
y0	
xleng	*Length of x-axis*
sxmax	*Maximum value of x-axis, must be > 0*
sxmin	*Minimum value of x-axis, must be > 0*
sizex	*Size of the numbers*
ticup	*If > 0, number tics are up, in +x direction*
labtop	*If > 0, place numbers and title above the x-axis*
dopow	*If > 0, annotate with numbers*
lx	*Number of characters in title string*
str	*Title string*

---

dnlinx

Plot and annotate the x-axis. The numbers decrease to the right.

```
void dnlinx(float x0,float y0,float xleng,float xmax,float xmin,
            float sizex,int ticup,int labtop,int dopow,
            int lx, char *str);
```

x0	*Coordinates of left side of axis*
y0	
xleng	*Length of x-axis*
xmax	*Maximum value of x-axis*
xmin	*Minimum value of x-axis*
sizex	*Size of the numbers*
ticup	*If > 0, number tics are up, in +x direction*
labtop	*If > 0, place numbers and title above the x-axis*
dopow	*If > 0, annotate with numbers*
lx	*Number of characters in title string*

str *Title string*

---

dolinx

Plot and annotate the x-axis. The numbers increase to the right.

```
void dolinx(float x0,float y0,float xleng,float xmax,float xmin,
            float sizex,int ticup,int labtop,int dopow,
            int lx, char *str);
```

x0 *Coordinates of left side of axis*
y0
xleng *Length of x-axis*
xmax *Maximum value of x-axis*
xmin *Minimum value of x-axis*
sizex *Size of the numbers*
ticup *If > 0, number tics are up, in +x direction*
labtop *If > 0, place numbers and title above the x-axis*
dopow *If > 0, annotate with numbers*
lx *Number of characters in title string*
str *Title string*

---

dolnx

Plot linear y-axis. Numbers increase from top to bottom.

```
void dolnx(float x0,float y0,float xleng,float xmax,float xmin,
        float sizex,int ticup,int labtop,int dopow,
            int llx, char *titlex, int doasis);
```

x0 *Coordinates of left side of axis*
y0
xleng *Length of x-axis*
xmax *Maximum value of x-axis*
xmin *Minimum value of x-axis*
sizex *Size of the numbers*
ticup *If > 0, number tics are up, in +x direction*
labtop *If > 0, place numbers and title above the x-axis*
dopow *If > 0, annotate with numbers*
lx *Number of characters in title string*
str *Title string*
doasis *If > 0 this implements dolinx, = 0 implements dnlinx*

dnliny

```
void dnliny(float x0,float y0,float yleng,float ymax,float ymin,
            float sizey,int ticlft,int lablft,int dopow,
            int ly, char *str);
```

x0	*Coordinates of bottom of axis*
y0	
yleng	*Length of y-axis*
ymax	*Maximum value of y-axis*
ymin	*Minimum value of y-axis*
sizey	*Size of numbers*
ticlft	*If > 0, number tics are to the left in y-direction*
lablft	*If > 0, place numbers to left of Y-axis*
dopow	*If > 0, annotate with numbers*
ly	*Number of characters in title*
titley	*Title string*

doliny

Plot linear y-axis. Numbers increase from bottom to top.

```
void doliny(float x0,float y0,float yleng,float ymax,float ymin,
            float sizey,int ticlft,int lablft,int dopow,
            int ly, char *str);
```

x0	*Coordinates of bottom of axis*
y0	
yleng	*Length of y-axis*
ymax	*Maximum value of y-axis*
ymin	*Minimum value of y-axis*
sizey	*Size of numbers*
ticlft	*If > 0, number tics are to the left in y-direction*
lablft	*If > 0, place numbers to left of Y-axis*
dopow	*If > 0, annotate with numbers*
ly	*Number of characters in title*
titley	*Title string*

dolny

This is the general purpose routine that implements the **doliny** and **dnliny** calls.

```
void dolny(float x0,float y0,float yleng,float ymax,float ymin,
        float sizey,int ticlft,int lablft,int dopow,
            int ly, char *titley, int doasis);
```

x0	*Coordinates of bottom of axis*
y0	
yleng	*Length of y-axis*
ymax	*Maximum value of y-axis*
ymin	*Minimum value of y-axis*
sizey	*Size of numbers*
ticlft	*If > 0, number tics are to the left in y-direction*
lablft	*If > 0, place numbers to left of Y-axis*
dopow	*If > 0, annotate with numbers*
ly	*Number of characters in title*
titley	*Title string*
doasis	*If > 0 implement doliny, = 0 implement dnliny*

fillit

Fill a solid symbol.

```
void fillit(char *cmd, float rad, float x0, float y0);
```

cmd	*"TR" for triangle*
	"SQ" for square
	"HX" for hexagon
	"CI" for circle
	"DI" for diamond
rad	*Radius of inscribed circle*
x0	*x-coordinate of center of symbol*
y0	*y-coordinate of center of symbol*

curvit

Draw a specified polygon shape. when combined with *fillit* permits solid symbol with outline set off by another color.

```
void curvit(char *cmd, float rad, float x0, float y0);
```

cmd	*"TR" for triangle*
	"SQ" for square
	"HX" for hexagon
	"CI" for circle
	"DI" for diamond
rad	*Radius of inscribed circle*
x0	*x-coordinate of center of symbol*
y0	*y-coordinate of center of symbol*

drawcv

Use internally to draw a curve. Since this is not a polygon, the last point is not connected to the first. If you require a polygon, add the coordinates of the first poiint to the end of the list.

```
void drawcv(int jj, float xval[], float yval[]);
```

jj	*Number of points in line segment*
xval	*Array of x-coordinates*
yval	*Array of y-coordinates*

rclip

Given a rectangular clip region defined by the corners *(xmin,ymin)* and *(xmax, ymax)*, determine whether a given line segment defined by *(x0,yy0)* and *(x1,yy1)* can be plotted *(*iplt > 0)*, and if so, return the visible coordinates *(*xc1,*yc1)* and *(xc2, yc2)*. This is an implementation fo a classic line clipping algorithm.

```
void rclip(float xmin,float xmax,float ymin,float ymax,float *xc1,
           float *yc1,float *xc2,float *yc2, float x0,float yy0,
           float x1,float yy1,int *iplt)  ;
```

xmin	*Minimum x-value of bounding box*
ymin	*Minimum y-value of bounding box*
xmax	*Maximum x-value of bounding box*
ymax	*Maximum y-value of bounding box*

xc1	
yc1	*coordinates of one end of line segment if within the clip region*
xc2	
yc2	*coordinates of other end of line segment if within the clip region*
x0	
yy0	*coordinate of one end of desired line*
x1	
yy1	*coordinate of other end of desired line*
iplt	*If > 0, plot line segment is within the plot region*

---

gsubsc

This plots two strings horizontally, with the second being a subscript: **s1$_{s2}$**

```
void  gsubsc(float x,float y,float ht,char *s1,int n1,char *s2,
            int n2);
```

x	*x-coordinate of beginning of string*
y	*y-coordinate of beginning of string*
ht	*height of string*
s1	*First string*
n1	*Number of characters in first string*
s2	*Second string that forms a subscript*
n2	*Number of characters in second string*
	If n1 or n2 are negative, the Greek font is used

---

gsupsc

This plots two strings horizontally, with the second being a superscript: **s1^{s2}**

```
void  gsupsc(float x,float y,float ht,char *s1,int n1,char *s2,
            int n2);
```

x	*x-coordinate of beginning of string*
y	*y-coordinate of beginning of string*
ht	*height of string*
s1	*First string*
n1	*Number of characters in first string*
s2	*Second string that forms a subscript*
n2	*Number of characters in second string*
	If n1 or n2 are negative, the Greek font is used

gsubsup

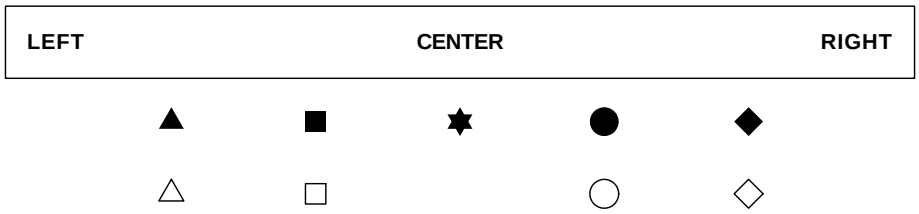
This plots three strings horizontally, with the second being a subscript and the third a super script: $\mathbf{s1}_{\mathbf{s2}}^{\mathbf{s3}}$

```
void  gsupsc(float x,float y,float ht,char *s1,int n1,char *s2,
           int n2);
```

x	*x-coordinate of beginning of string*
y	*y-coordinate of beginning of string*
ht	*height of string*
s1	*First string*
n1	*Number of characters in first string*
s2	*Second string that forms a subscript*
n2	*Number of characters in second string*
s3	*Second string that forms a subscript*
n3	*Number of characters in second string*
	If n1, n2 or n3 are negative, the Greek font is used

2. C Examples

The examples given here illustrate the use of these routines. The extract of the C code that creates the plots is given for each.




```
gbox(1.1,5.8,7.4,6.3);
gleft(1.25,6.0,0.10,"LEFT",0.0);
gright(7.25,6.0,0.10,"RIGHT",0.0);
gcent(4.25,6.0,0.10,"CENTER",0.0);
fillit("TR",0.1,2.25,5.5);
fillit("SQ",0.1,3.25,5.5);
fillit("HX",0.1,4.25,5.5);
fillit("CI",0.1,5.25,5.5);
fillit("DI",0.1,6.25,5.5);
curvit("TR",0.1,2.25,5.0);
curvit("SQ",0.1,3.25,5.0);
curvit("CI",0.1,5.25,5.0);
curvit("DI",0.1,6.25,5.0);
```

Fig 6.1 String and symbol routines

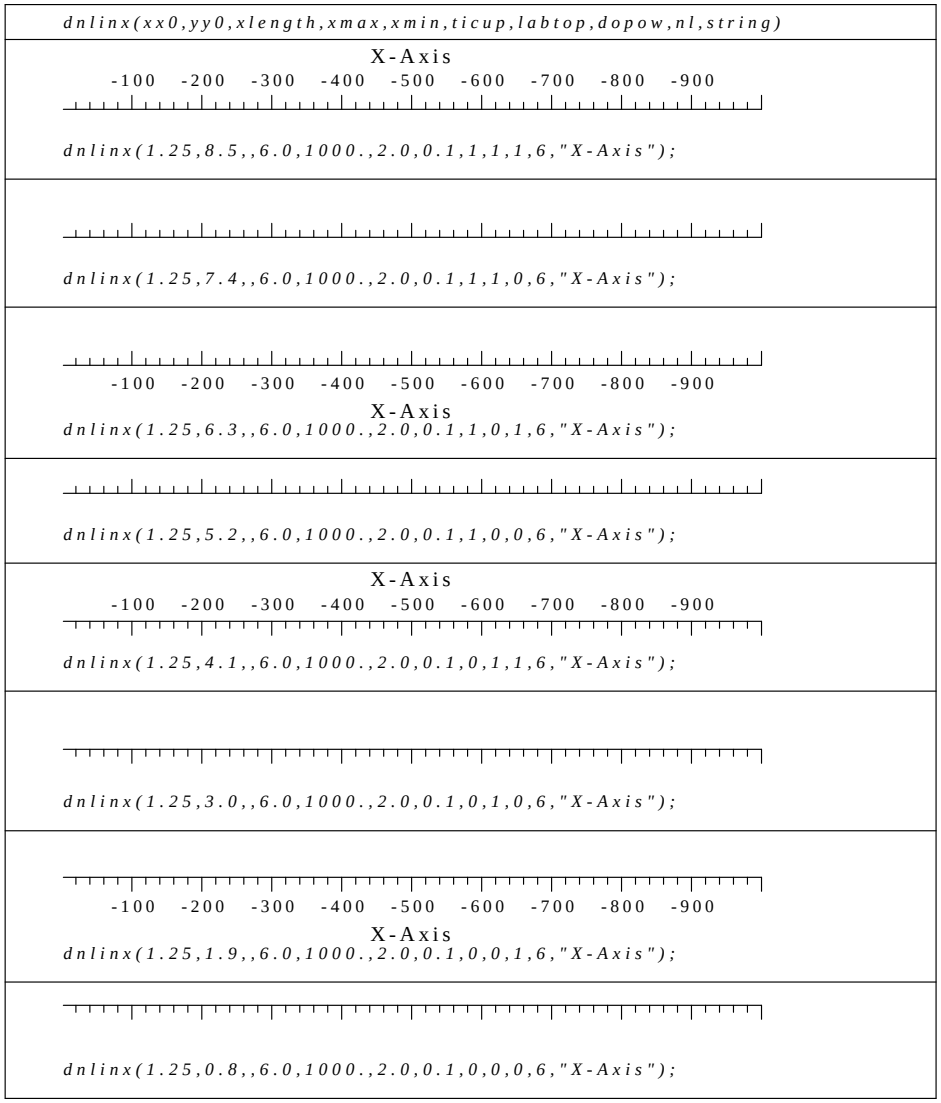
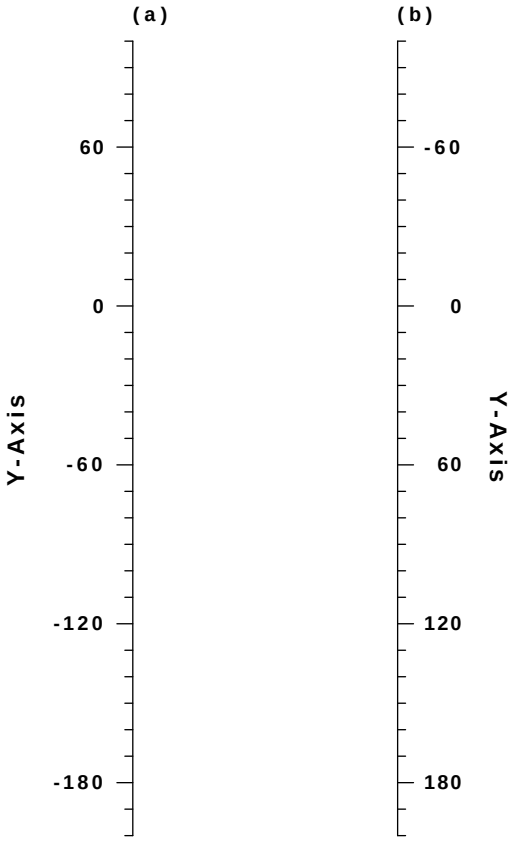


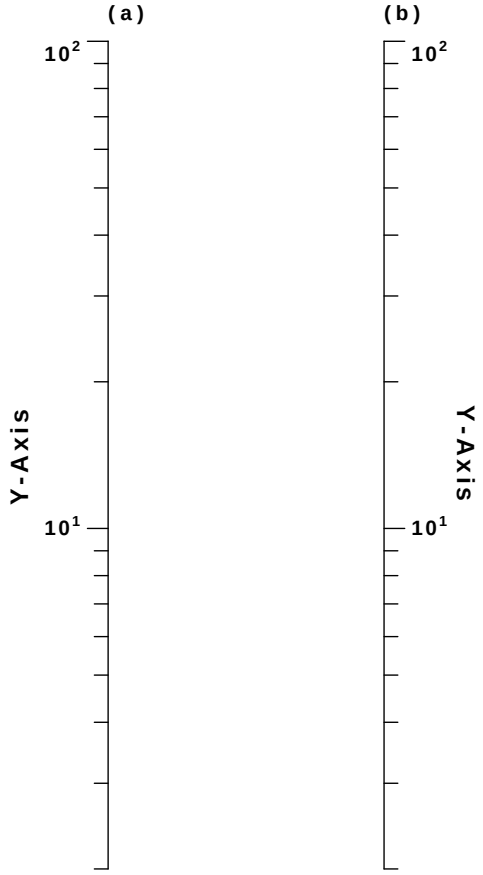
Fig 6.2 Example of the **`dnlinx`** routines.




```
(a): doliny(2.0,1.25,6.0,100.,-200.0,0.1,TRUE,TRUE,TRUE,6,"Y-Axis");

(b): dnliny(4.0,1.25,6.0,100.,-200.0,0.1,FALSE,FALSE,TRUE,6,"Y-Axis");
```

Fig 6.3 Example of the **`doliny`** and **`dnliny`** routines.




```
(a): dology(2.0,1.25,6.0,100.,2.0,0.1,TRUE,TRUE,TRUE,6,"Y-Axis");

(b): dology(4.0,1.25,6.0,100.,2.0,0.1,FALSE,FALSE,TRUE,6,"Y-Axis");
```

Fig 6.4 Example of the **`dology`** routine.

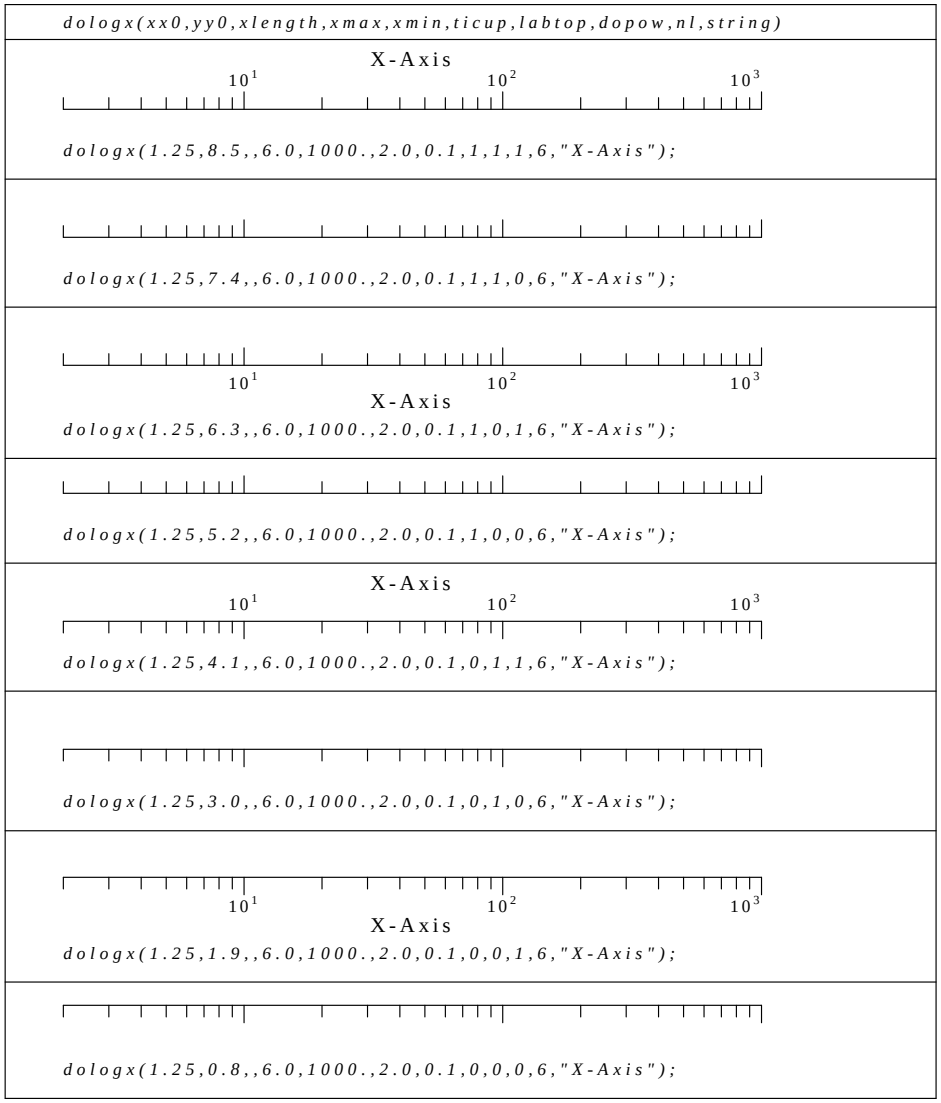


Fig 6.5 Example of the **`dologx`** routine.

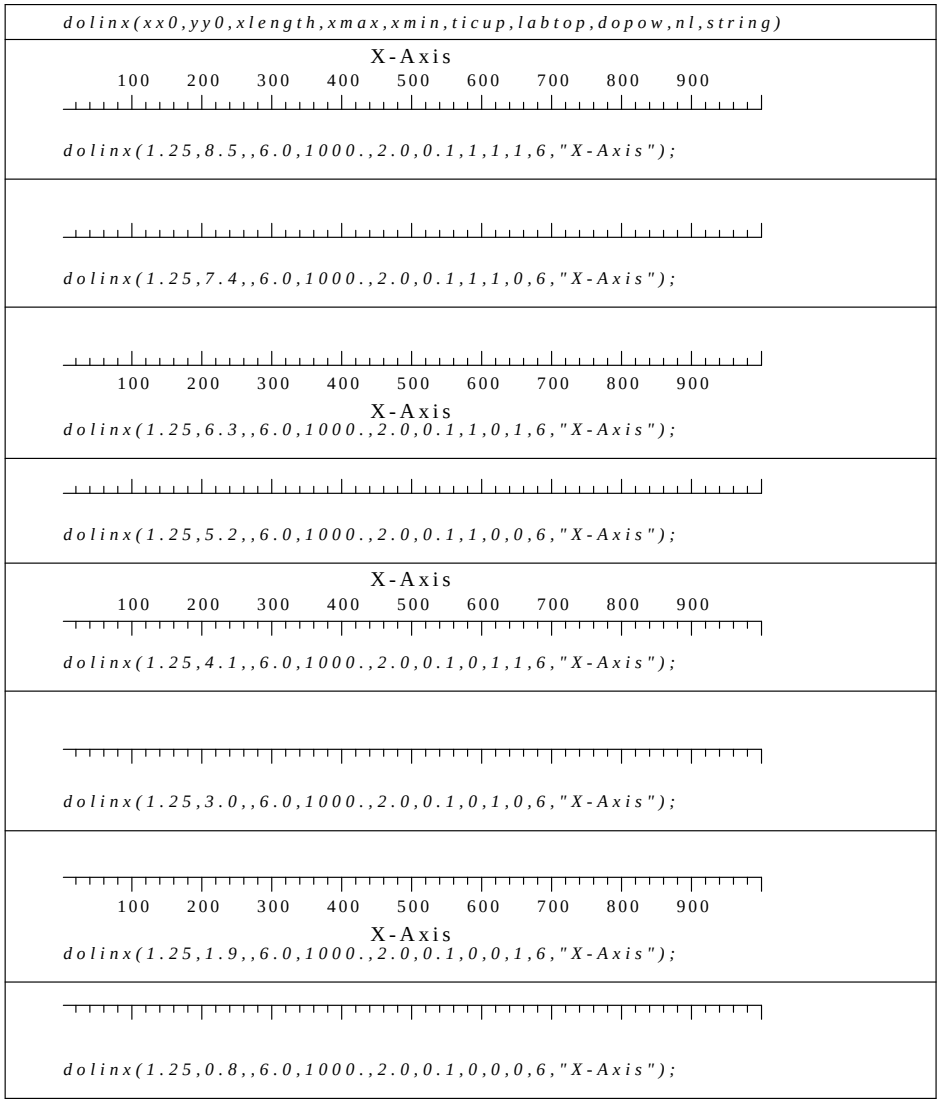


Fig 6.6 Example of the **`dolinx`** routine.

$P_n(x)$	`gsubsc(1.0,7.0,0.10,"P",1,"n",1);`
$P_\nu(x)$	`gsubsc(1.0,7.0,0.10,"P",1,"n",-1);`
$\Pi_\nu(x)$	`gsubsc(1.0,7.0,0.10,"P",-1,"n",-1);`
$G^a(x)$	`gsupsc(1.0,7.0,0.10,"G",1,"a",1);`
$G^\alpha(x)$	`gsupsc(1.0,7.0,0.10,"G",1,"a",-1);`
$\Gamma^\alpha(x)$	`gsupsc(1.0,7.0,0.10,"G",-1,"a",-1);`
$G_a^{bg}(x)$	`gsubsup(1.0,7.0,0.10,"G",1,"a",1,"bg", 2);`
$G_\alpha^{bg}(x)$	`gsubsup(1.0,7.0,0.10,"G",1,"a",-1,"bg", 2);`
$G_\alpha^{\beta\gamma}(x)$	`gsubsup(1.0,7.0,0.10,"G",1,"a",-1,"bg",-2);`
$\Gamma_\alpha^{\beta\gamma}(x)$	`gsubsup(1.0,7.0,0.10,"G",-1,"a",-1,"bg",-2);`

Fig 6.7 Example of the **gsubsc**, **0gsupsc** and **gsubsup** routines.

3. FORTRAN Routines

---

gbox

Draw a box

subroutine gbox(xl, yl, xh, yh)
real xl *x-coordinate of the lower left corner*
real yl *y-coordinate of the lower left corner*
real xh *x-coordinate of the upper right corner*
real yh *y-coordinate of the upper right corner*

---

gcent

Draw text centered at the coordinates *(xx, yy)*.

subroutine gcent(xx, yy, ht,string, angle)
real xx *x-coordinate of the lower left corner of the first character*
real yy *y-coordinate of the lower left corner of the first character*

real ht *height if the character*
character string*(*) *character string to tbe plotted*
real angle *angle at which the string is plotted measured from x-axis*

---

gright

Draw text right justified at the coordinates *(xx, yy)*.

subroutine gright(xx, yy, ht,string, angle)
real xx *x-coordinate of the lower left corner of the first character*
real yy *y-coordinate of the lower left corner of the first character*
real ht *height if the character*
character string*(*) *character string to tbe plotted*
real angle *angle at which the string is plotted measured from x-axis*

---

gleft

Draw text left justfied at the coordinates *(xx, yy)*

subroutine gleft(xx, yy, ht,string, angle)
real xx *x-coordinate of the lower left corner of the first character*
real yy *y-coordinate of the lower left corner of the first character*
real ht *height if the character*
character string*(*) *character string to tbe plotted*
real angle *angle at which the string is plotted measured from x-axis*

---

dology

Plot a logarithmic y-axis. This differs from the **algaxe** routine in that the starting and end values need not be exact powers of 10.

subroutine dology(x0, y0, yleng, ymax, ymin,
sizey, ticlft, lablft, dopow, ly, str)
real x0 *Coordinates of bottom of axis*
real y0
real yleng *Length of y-axis*
real ymax *Maximum value of y-axis, must be > 0*
real ymin *Minimum value of y-axis, must be > 0*

real sizey *Size of numbers*
logical ticlft*If .true., number tics are to the left in y-direction*
logical lablft*If .true., place numbers to left of Y-axis*
logical dopow*If .true., annotate with numbers*
integer ly *Number of characters in title*
character titley*(*)*Title string*

---

dologx

Plot a logarithmic x-axis. This differs from the **algaxe** routine in that the starting and end values need not be exact powers of 10.

```
subroutine dologx(float x0,float y0,float xleng,float sxmax,float sxmin,
           float sizex,int ticup,int labtop,int dopow,
           int lx, char *str)
```
real x0 *Coordinates of left side of axis*
real y0
real xleng *Length of x-axis*
real sxmax *Maximum value of x-axis, must be > 0*
real sxmin *Minimum value of x-axis, must be > 0*
real sizex *Size of the numbers*
logical ticup*If > 0, number tics are up, in +x direction*
logical labtop*If > 0, place numbers and title above the x-axis*
logical dopow*If > 0, annotate with numbers*
integer lx *Number of characters in title string*
character str*(*)*Title string*

---

dnlinx

```
subroutine dnlinx( x0, y0, xleng, xmax, xmin,
            sizex,ticup,labtop,dopow, lx, str)
```
real x0 *Coordinates of left side of axis*
real y0
real xleng *Length of x-axis*
real xmax *Maximum value of x-axis*
real xmin *Minimum value of x-axis*
real sizex *Size of the numbers*
logical ticup*If .true., number tics are up, in +x direction*

```
logical labtop   If .true., place numbers and title above the x-axis
logical dopow    If .true., annotate with numbers
integer lx       Number of characters in title string
character str*(*) Title string
```

---

dolinx

```
subroutine dolinx(x0,y0,xleng,xmax,xmin,
                  sizex,ticup,labtop,dopow, lx, str)
real x0          Coordinates of left side of axis
real y0
real xleng       Length of x-axis
real xmax        Maximum value of x-axis
real xmin        Minimum value of x-axis
real sizex       Size of the numbers
logical ticup    If .true., number tics are up, in +x direction
logical labtop   If .true., place numbers and title above the x-axis
logical dopow    If .true., annotate with numbers
integer lx       Number of characters in title string
character str*(*) Title string
```

---

dolnx

```
subroutine dolnx(x0,y0,xleng,xmax,xmin,
        sizex,ticup,labtop,dopow,
                  llx, titlex, doasis)
real x0          Coordinates of left side of axis
real y0
real xleng       Length of x-axis
real xmax        Maximum value of x-axis
real xmin        Minimum value of x-axis
real sizex       Size of the numbers
logical ticup    If .true., number tics are up, in +x direction
logical labtop   If .true., place numbers and title above the x-axis
logical dopow    If .true., annotate with numbers
integer lx       Number of characters in title string
character str*(*) Title string
```

logical doasis*If .true. implement dolinx, else dnlinx*

---

dnliny

```
subroutine dnliny(x0,y0,yleng,ymax,ymin, sizey,ticlft,lablft,dopow,
          ly, str)
real x0      Coordinates of bottom of axis
real y0
real yleng   Length of y-axis
real ymax    Maximum value of y-axis
real ymin    Minimum value of y-axis
real sizey   Size of numbers
logical ticlft If .true., number tics are to the left in y-direction
logical lablft If .true., place numbers to left of Y-axis
logical dopow If .true., annotate with numbers
integer ly   Number of characters in title
character titley(*) Title string
```

---

doliny

```
subroutine doliny( x0, y0, yleng, ymax, ymin, sizey,ticlft,lablft,dopow,
          ly, str)
real x0      Coordinates of bottom of axis
real y0
real yleng   Length of y-axis
real ymax    Maximum value of y-axis
real ymin    Minimum value of y-axis
real sizey   Size of numbers
logical ticlft If .true., number tics are to the left in y-direction
logical lablft If .true., place numbers to left of Y-axis
logical dopow If .true., annotate with numbers
integer ly   Number of characters in title
character titley(*) Title string
```

dolny

This is the general purpose routine that implements the **doliny** and **dnliny** calls.

```
subroutine dolny( x0, y0, yleng, ymax, ymin, sizey,ticlft,lablft,dopow,
           lly, titley, doasis)
real x0       Coordinates of bottom of axis
real y0
real yleng    Length of y-axis
real ymax     Maximum value of y-axis
real ymin     Minimum value of y-axis
real sizey    Size of numbers
logical ticlftIf .true., number tics are to the left in y-direction
logical lablftIf .true., place numbers to left of Y-axis
logical dopowIf .true., annotate with numbers
integer ly    Number of characters in title
character titley(*)Title string
logical doasisIf .true. implement doliny, else dnliny
```

fillit

Fill a solid symbol.

```
subroutine fillit(*cmd,  rad,  x0,  y0)
cmd           'TR' for triangle
              'SQ' for square
              'HX' for hexagon
              'CI' for circle
              'DI' for diamond
real rad      Radius of inscribed circle
real x0       x-coordinate of center of symbol
real y0       y-coordinate of center of symbol
```

curvit

Draw a specified polygon shape. when combined with *fillit* permits solid symbol with outline set off by another color.

```
subroutine curvit(cmd,  rad, x0, y0)
character cmd*(*)'TR' for triangle
                 'SQ' for square
                 'HX' for hexagon
                 'CI' for circle
                 'DI' for diamond
real rad         Radius of inscribed circle
real x0          x-coordinate of center of symbol
real y0          y-coordinate of center of symbol
```

drawcv

Use internally to draw a curve. Since this is not a polygon, the last point is not connected to the first. If you require a polygon, add the coordinates of the first poiint to the end of the list.

```
subroutine drawcv(jj, xval,  yval)
integer jj   Number of points in line segment
real xval    Array of x-coordinates
real yval    Array of y-coordinates
```

rclip

Given a rectangular clip region defined by the corners *(xmin,ymin)* and *(xmax, ymax)*, determine whether a given line segment defined by *(x0,yy0)* and *(x1,yy1)* can be plotted *(*iplt > 0)*, and if so, return the visible coordinates *(*xc1,*yc1)* and *(xc2, yc2)*. This is an implementation fo a classic line clipping algorithm.

```
subroutine rclip( xmin, xmax, ymin, ymax, xc1, yc1, xc2, yc2,  x0, yy0, x1, yy1,iplt)
real xmin    Minimum x-value of bounding box
real ymin    Minimum y-value of bounding box
real xmax    Maximum x-value of bounding box
```

real ymax *Maximum y-value of bounding box*
real xc1
real yc1 *coordinates of one end of line segment if within the clip region*
real xc2
real yc2 *coordinates of other end of line segment if within the clip region*
real x0
real yy0 *coordinate of one end of desired line*
real x1
real yy1 *coordinate of other end of desired line*
iplt *If > 0, plot line segment is within the plot region*

---

gsubsc

This plots two strings horizontally, with the second being a subscript: **s1$_{s2}$**

subroutine gsubsc(x,y,ht,s1,n1,s2,n2)
real x *x-coordinate of beginning of string*
real y *y-coordinate of beginning of string*
real ht *height of string*
character s1*(*) *First string*
integer n1 *Number of characters in first string*
character s2*(*) *Second string that forms a subscript*
integer n2 *Number of characters in second string*
If n1 or n2 are negative, the Greek font is used

---

gsupsc

This plots two strings horizontally, with the second being a superscript: **s1^{s2}**

subroutine gsupsc(x,y,ht,s1,n1,s2,n2)
real x *x-coordinate of beginning of string*
real y *y-coordinate of beginning of string*
real ht *height of string*
character s1*(*) *First string*
integer n1 *Number of characters in first string*
character s2*(*) *Second string that forms a subscript*
integer n2 *Number of characters in second string*
If n1 or n2 are negative, the Greek font is used

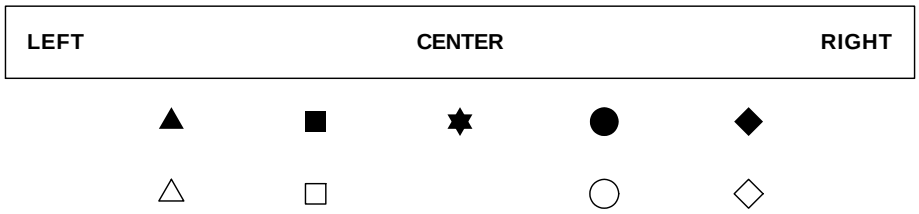
gsubsup

This plots three strings horizontally, with the second being a subscript and the third a super script: $\mathbf{s1}_{\mathbf{s2}}^{\mathbf{s3}}$

```
subroutine  gsupsc(x,y,ht,s1,n1,s2,n2)
real x      x-coordinate of beginning of string
real y      y-coordinate of beginning of string
real ht     height of string
character s1*(*)      First string
integer n1      Number of characters in first string
character s2*(*)      Second string that forms a subscript
integer n2      Number of characters in second string
character s3*(*)      Second string that forms a subscript
integer n3      Number of characters in second string
        If n1, n2 or n3 are negative, the Greek font is used
```

4. FORTRAN Examples

The examples given here illustrate the use of these routines. The extract of the FORTRAN code that creates the plots is given for each.




```
call gbox(1.1,5.8,7.4,6.3)
call gleft(1.25,6.0,0.10,'LEFT',0.0)
call gright(7.25,6.0,0.10,'RIGHT',0.0)
call gcent(4.25,6.0,0.10,'CENTER',0.0)
call fillit('TR',0.1,2.25,5.5)
call fillit('SQ',0.1,3.25,5.5)
call fillit('HX',0.1,4.25,5.5)
call fillit('CI',0.1,5.25,5.5)
call fillit('DI',0.1,6.25,5.5)
call curvit('TR',0.1,2.25,5.0)
call curvit('SQ',0.1,3.25,5.0)
call curvit('CI',0.1,5.25,5.0)
call curvit('DI',0.1,6.25,5.0)
```

Fig 6.8 String and symbol routines

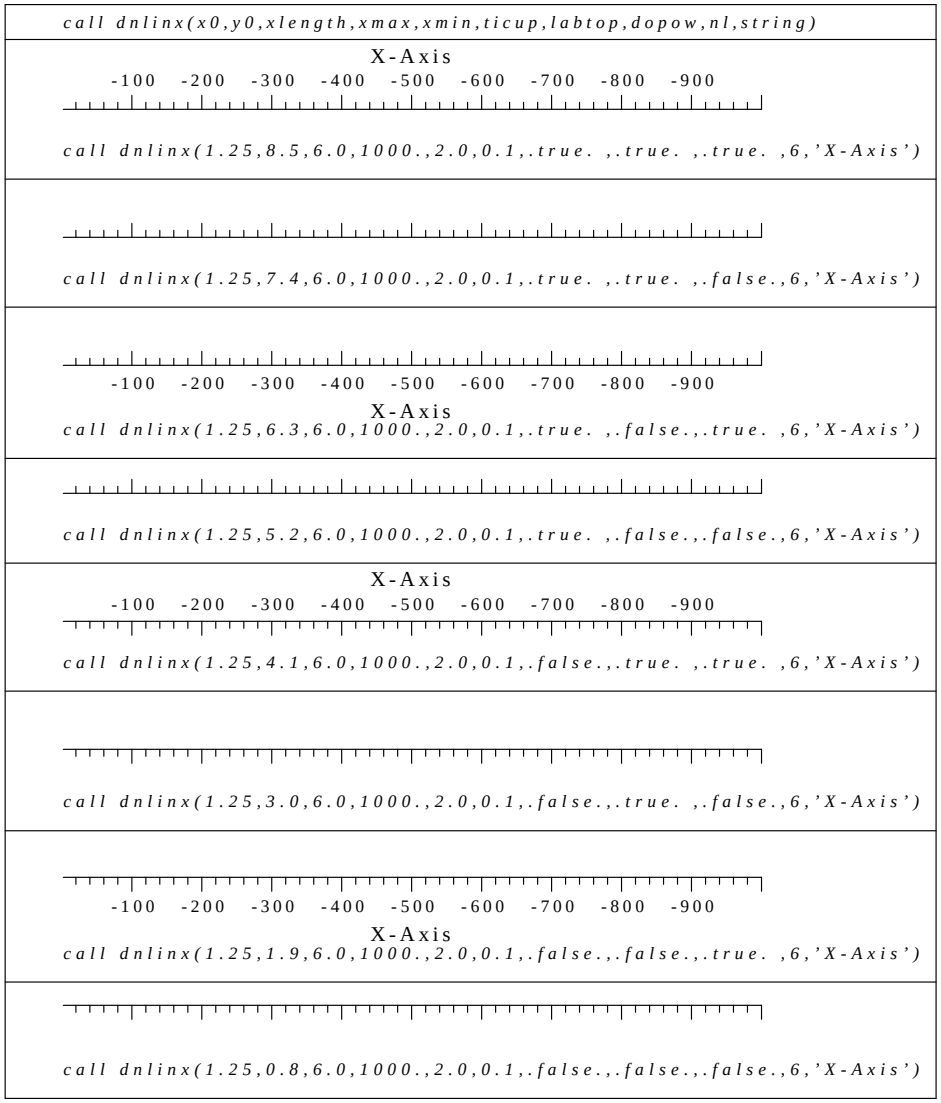
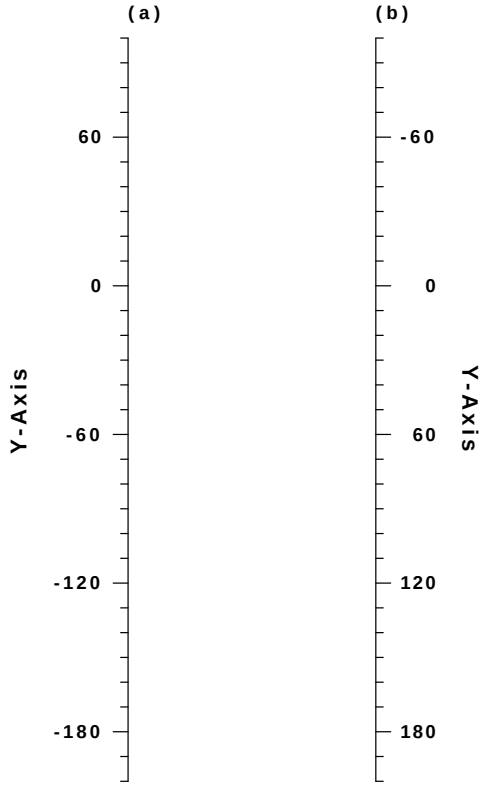


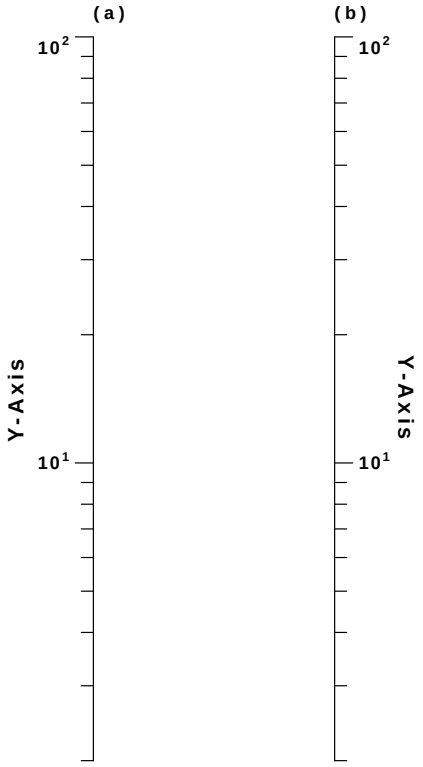
Fig 6.9 Example of the **`dnlinx`** routines.




```
(a): doliny(2.0,1.25,6.0,100.,-200.0,0.1,.true.,.true.,.true.,6,'Y-Axis')

(b): dnliny(4.0,1.25,6.0,100.,-200.0,0.1,.false.,.false.,.true.,6,'Y-Axis')
```

Fig 6.10 Example of the **`doliny`** and **`dnliny`** routines.




```
(a): call dology(2.0,1.25,6.0,100.,2.0,0.1,.true.,.true.,.true.,6,'Y-Axis')

(b): call dology(4.0,1.25,6.0,100.,2.0,0.1,.false.,.false.,.true.,6,'Y-Axis')
```

Fig 6.11 Example of the **`dology`** routine.

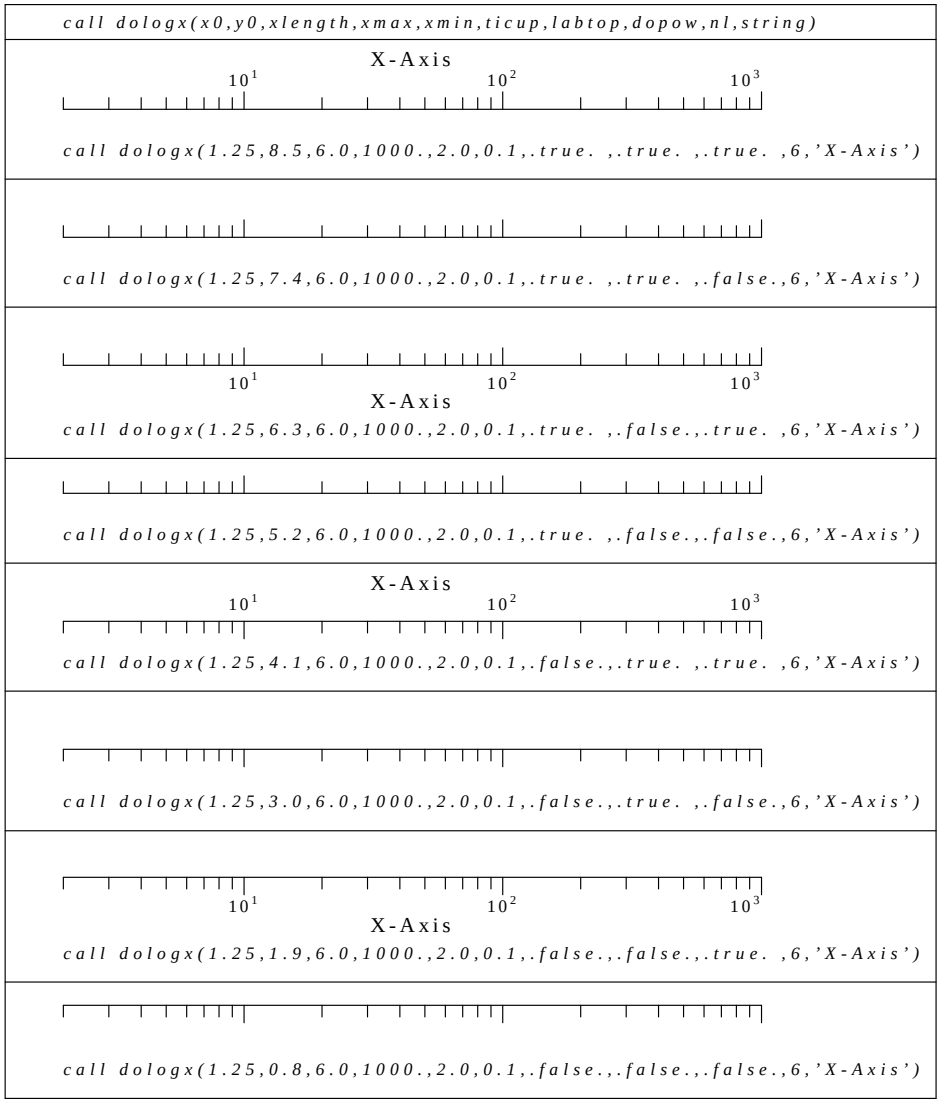


Fig 6.12 Example of the **`dologx`** routine.

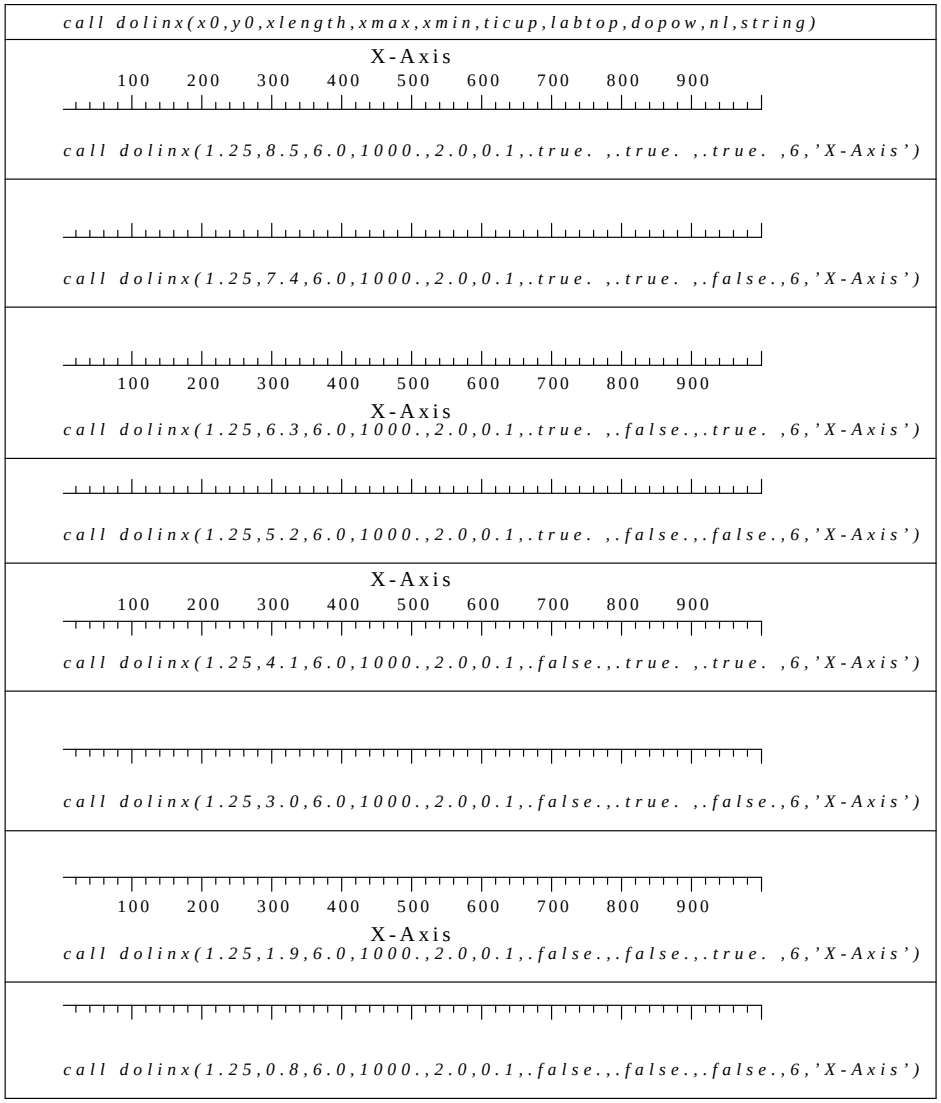


Fig 6.13 Example of the **`dolinx`** routine.

$P_n(x)$	`call gsubsc(1.0,7.0,0.10,'P',1,'n',1)`
$P_\nu(x)$	`call gsubsc(1.0,7.0,0.10,'P',1,'n',-1)`
$\Pi_\nu(x)$	`call gsubsc(1.0,7.0,0.10,'P',-1,'n',-1)`
$G^a(x)$	`call gsupsc(1.0,7.0,0.10,'G',1,'a',1)`
$G^\alpha(x)$	`call gsupsc(1.0,7.0,0.10,'G',1,'a',-1)`
$\Gamma^\alpha(x)$	`call gsupsc(1.0,7.0,0.10,'G',-1,'a',-1)`
$G_a^{bg}(x)$	`call gsubsup(1.0,7.0,0.10,'G',1,'a',1,'bg', 2)`
$G_\alpha^{bg}(x)$	`call gsubsup(1.0,7.0,0.10,'G',1,'a',-1,'bg', 2)`
$G_\alpha^{\beta\gamma}(x)$	`call gsubsup(1.0,7.0,0.10,'G',1,'a',-1,'bg',-2)`
$\Gamma_\alpha^{\beta\gamma}(x)$	`call gsubsup(1.0,7.0,0.10,'G',-1,'a',-1,'bg',-2)`

Fig 6.14 Example of the **gsubsc**, **0gsupsc** and **gsubsup** routines.

CHAPTER 7
UTILITY PROGRAMS

Given the programming tools in *CALPLOT*, interesting things can be done. The program **reframe** is written at the device driver level and does not use the library routines directly, but does permit repositioning of plots generated using *CALPLOT*. Another program **CAL** is a partially developed interface to *CALPLOT* that permits simple interactive plotting without writing new programs. It is used in conjunction with **reframe** to append legends to existing figures.

1. reframe

The program **reframe** is described in the Appendix. This program permits manipulation of the binary plot files in order to reposition the plot on the page, to extract a given page or to extract a portion of the plot.

2. CAL

A need arose for a simple interactive program to position simple plot commands on a screen. For example, one might wish to place the identifiers **(a)**, **(b)**, etc., on some existing plots in order to produce a finished plot for publication. This could be done by writing a specific program for each modification, but this becomes tedious. A general purpose program is of greater value. The program **CAL** provides partial access to the *CALPLOT* library of commands. It gets input from the standard input. If invoked with the *-V* flag, syntax is provided, which is listed here:

```
FACTOR:scaling_factor
SYMBOL:x,y,ht,string,angle,nchar
CIRCLE:rad,x0,y0
SFILL:string_type,height,x0,y0 where type = SQuare, CIrcle
DIamond, TRiangle, HXagon
ARC:rad,x0,y0,ang1,ang2
HELP: (show this menu)
CENTER:x,y,ht,string,angle
LEFT:x,y,ht,string,angle
RIGHT:x,y,ht,string,angle
NUMBER:x,y,ht,string,angle,nchar
NEWPEN:ipen
```

```
PLOTD:x,y,ipat,xlen
SHADER:x1,y1,x2,y2,ipatx,ipaty,xlen,ylen
PLOT:x,y,ipen
FRAME: (start new plot frame)
GWIDTH:width
GFONT:ifont
GUNIT:in or cm (string)
PEND:
LINE:xl,yl,xu,yu
BOX:xl,yl,xu,yu,string,ht
COMMENT: comment for CAL.plt in quotes
ARROW:xs,ys,xe,ye,bold(TF)
SUBSC:x,y,ht,s1,n1,s2,n2: - n1,n2 or neg = font 4
SUPSC:x,y,ht,s1,n1,s2,n2: - n1,n2  neg = font 4
SUBSUP:x,y,ht,s1,n1,s2,n2,s3,n3: - n1,n2,n3  neg = font 4
AXES3:x0,y0,xl,yl,zl,ang1,ang2,ang3,t1,s1,t2,s2,t3,s3,cmd
AXES3SU:x0,y0,xl,yl,zl,ang1,ang2,ang3,t1,s1,u1,t2,s2,u2,t3,s3,u3,cmd
ARRAY: enter quoted file name containing x,y pairs
Enter command
```

Whenever the syntax requires the input of a string, this string must be enclosed within single quotes, since the program was written in FORTRAN. A sample program would be

```
NEWPEN
2
LEFT
1.0 1.0 0.5 'This is a test' 0.0
LINE
2.0 3.0 2.5 3.5
PEND
```

Two examples are given here. The first, Figure 7.1 shows the result of the following input to the program CAL. The plot was used to indicate two possible interpretations of a data set.

```
PLOT
3.0 6.0 3
PLOT
7.0 6.0 2
PLOT
5.0 6.0 3
PLOT
5.0 5.5 2
PLOT
3.0 5.5 3
```

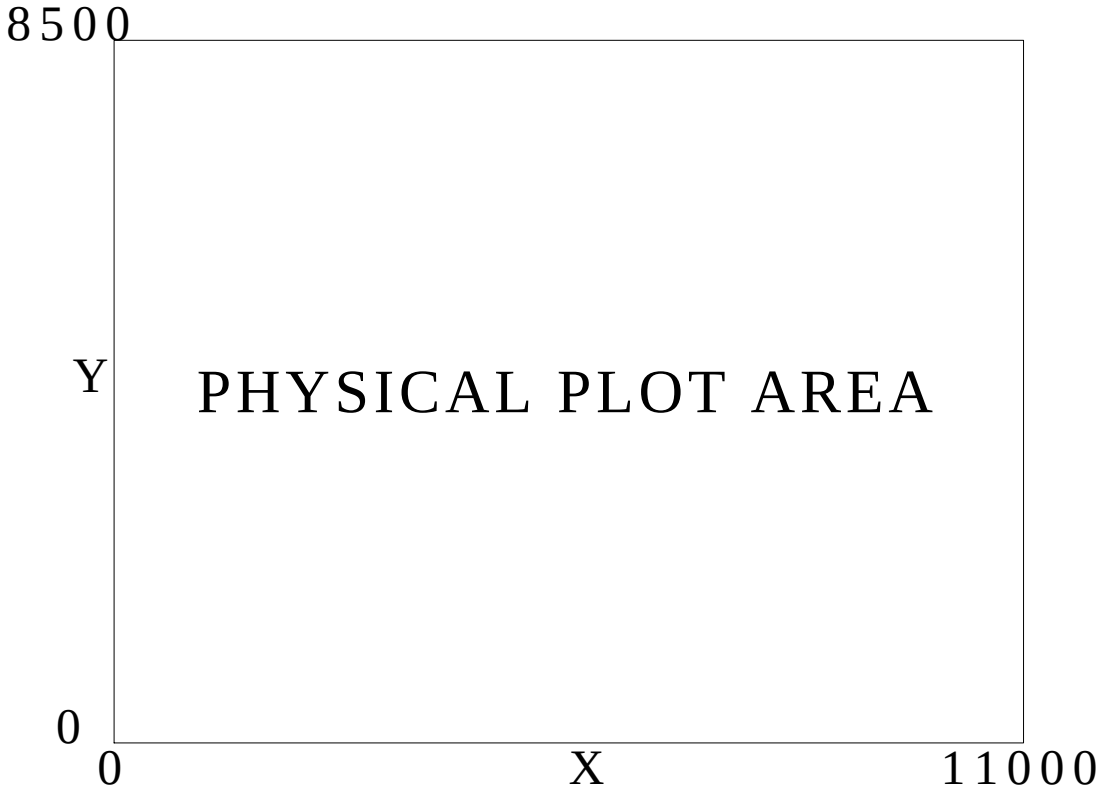


Fig. 7.1. Output of first example of using CAL of PLOTreframe2

```
PLOT
7.0 5.5 2
PLOT
3.0 5.0 3
PLOT
7.0 5.0 2
NEWPEN
1090
SHADER
5.0 5.5 7.0 6.0 0 0 0.1 0.1
NEWPEN
1050
SHADER
3.0 5.0 7.0 5.5 0 0 0.1 0.1
NEWPEN
1
SYMBOL
4.88 4.5 0.20 'OR' 0.0 2
PLOT
```

```
3.0 4.0 3
PLOT
7.0 4.0 2
PLOT
3.0 3.5 3
PLOT
5.0 3.5 2
PLOT
5.0 3.0 2
PLOT
7.0 3.0 2
PLOT
3.0 2.5 3
PLOT
7.0 2.5 2
NEWPEN
1060
SHADER
3.0 2.5 5.0 3.5 0 0 0.1 0.1
SHADER
5.0 2.5 7.0 3.0 0 0 0.1 0.1
```

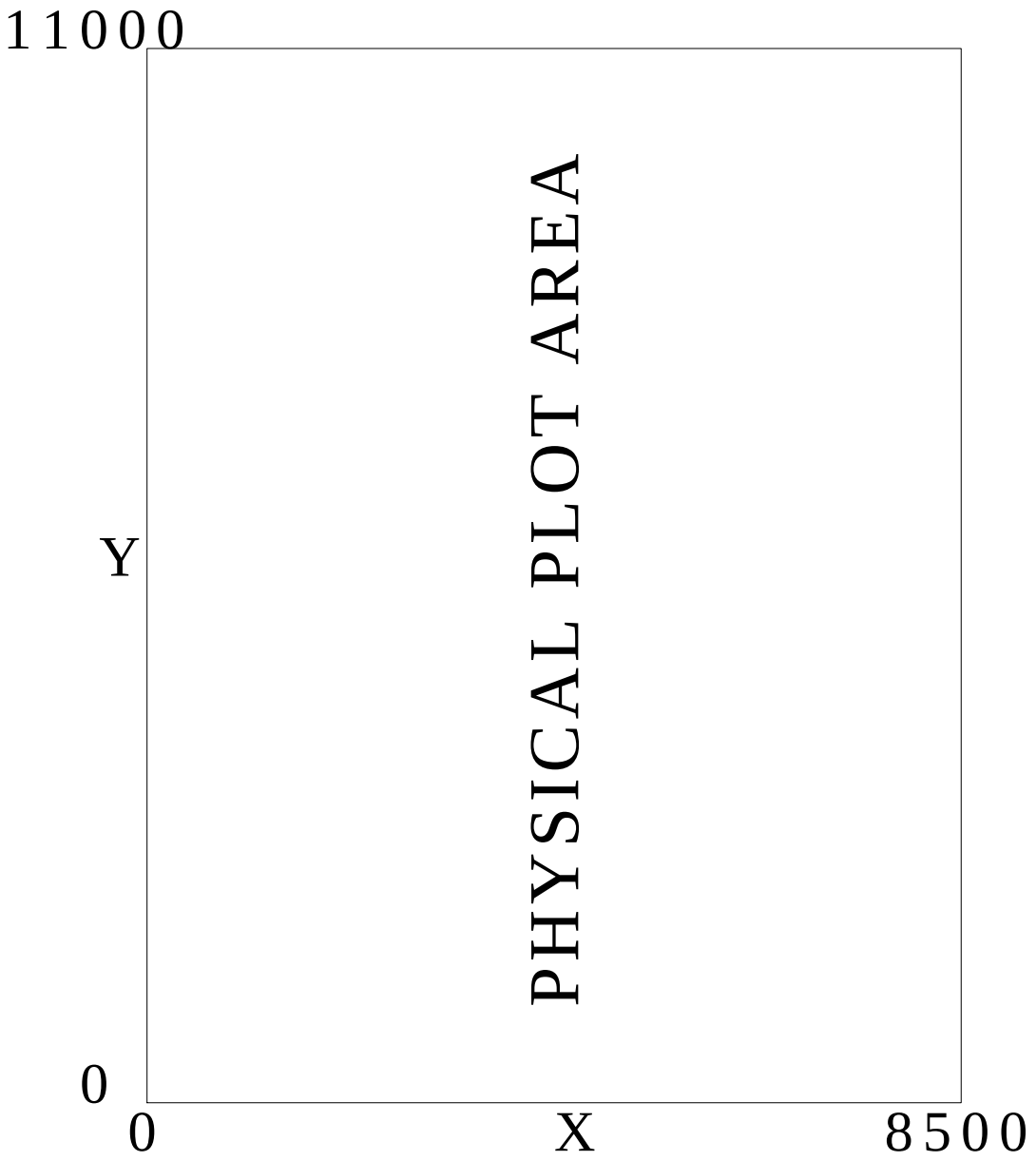
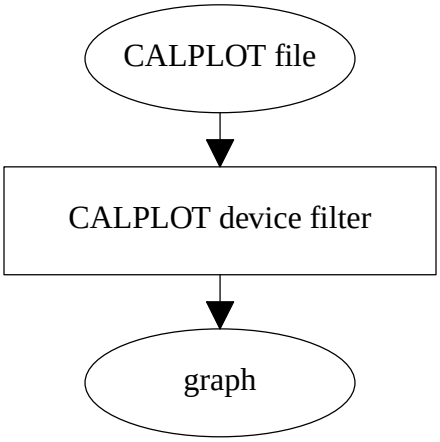


Fig. 7.2. Output of first example of using CAL of PLOTreframe2

APPENDIX A
CALPLOT GRAPHICS

1. Introduction

Computer Programs in Seismology is distributed with its own graphics package to make the installation of distributed software easier. Each non-interactive graphics program will create a binary, device-independent metafile of plotting commands, which must be converted for use by a specific hardware device. At the simplest level, the low level plotting commands are a sequence of pen up, pen move and pen down commands. Some of the early plotting devices supported were Calcomp mechanical plotters, Versatec electrostatic printers and Tektronix graphics terminals. Today graphic output is supported for X-Windows, PC Windows displays, and PostScript printers. Only a small subset of output devices are currently supported, primarily because of the existence of excellent conversion software; one example of which is *ghostscript* which converts PostScript to many devices.



If a CALPLOT device filter is named **plotdev**, one uses the program as follows:

`plotdev [options] < PLOTFILE` *for a screen device*

`plotdev [options] < PLOTFILE > temp_file` *(create temp file)*
`print temp_file` *(output to the actual printer)*

Some common options are

`-S`*scaling_factor*
: Multiply all plot moves by the *scaling_factor* (default 1.0)

`-R`
: Rotate the plot by 90°

`-F`*font*
: Change the default fault to number *font*. The default is Times Roman. A **`-F7`** will invoke bold Helvetica in PostScript.

Other commands are specific to the hardware device. A complete description of all supported devices is given in **CALPLOT(I)** of *Computer Programs in Seismology*.

2. PostScript Output

The program **plotnps** converts the binary CALPLOT file to PostScript. This program supports 128 unique colors in its palette. The output can also be in the form of Encapsulated PostScript, which is used to provide all graphics in this document.

Program control is through the command line:

plotnps *[flags]*, where the command flags are

-S*scalefac*
Scale all plot motions by this factor.

-P*pipe_process*
On UNIX/LINUX pipe the PostScript output through this process instead of sending through *stdout*

-R
Rotate the plot on the printed page. In effect the plot region is 8.5" wide and 11.0" high instead of 11.0" wide and 8.5" high.

-N
Turn off shading options for smaller PostScript file

-F*font*
Make the default font equal *font*

font	Font Used
0	Times-Roman
1	Times-Roman
2	Times-Italic
3	Times-Bold
4	Symbol (Greek)
5	Helvetica
6	Helvetica-Oblique
7	Helvetica-Bold
8	Symbol (Greek)
9	Courier
10	Courier-Oblique
11	Courier-Bold
12	Symbol (Greek)

-H30
-H60
Use a halftone density of 30 or 60 (default) dots per inch. The density of 30 produces larger dots, and may be of use when a figure must be reduced for publication.

-K
Show colors with a red -> green -> blue palette

-KR

Show colors with a red -> white -> blue pallette

-KB

Show colors with a blue -> white -> red pallette

-KW

Show colors, but whiten the spectrum.

-G

Show colors in grayscale.

The default action when neither **-G**, **-K** nor **-KW** are used is that shading is in gray, but all colored lines and text are black.

-B

assume the paper is 11 x 14 instead of 8.5 x 11

-L

assume the paper is 8.5 x 14 instead of 8.5 x 11

-A3

assume the paper is A3 instead of 8.5 x 11

-A4

assume the paper is A4 instead of 8.5 x 11

-W*min_linewidth*

Reset the minimum line width.

-EPS

Make the output an Encapsulated PostScript file.

-T*title*

Place the title string *title* in the lower left corner. Do not use spaces, or under UNIX/LINUX place string between quotes, e.g., **-T***"a test case"*

-h

-?

Online help

Standards

To be compatible with PostScript display software and with word processing software that permits inclusion of PostScript files, PostScript Document Structure Convention 3.0 (DSC 3.0) is followed.

Plotspace Mapping

The CALPLOT definition of axes is such that the X-axis is horizontal and the Y-axis is vertical. This is then mapped onto a printed page of dimension 8.5" x 11". In the default case the X-axis is mapped onto the long dimension of the paper. The plot space

on the paper is demonstrated in Figure 1.

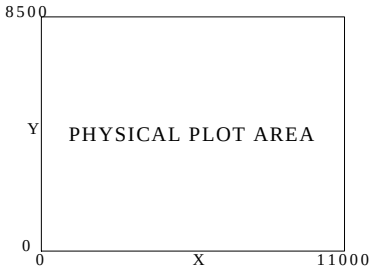


Fig. 1. Default mapping of CALPLOT plot space onto physical page.

The **-R** option rotates the mapping, such that the Y-axis is mapped onto the long dimension of the paper. The plot space on the paper is demonstrated in Figure 2.

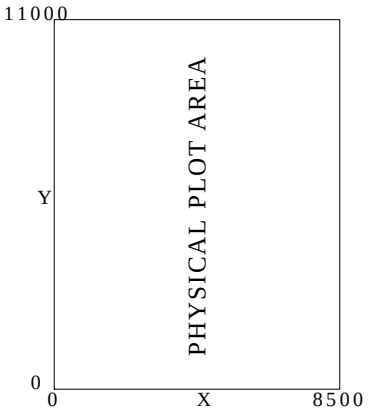


Fig. 2. Mapping of CALPLOT plot space onto physical page using *-R*.

Note that the CALPLOT plot space is mapped onto a rectangular page with no distortion of the unit lengths of the X- or Y-axes.

The PostScript plot space is assumed to be that the X-axis is horizontal with a length of 8.5" and the Y-axis is vertical with a length of 11.0". The use of the **-EPS** or **-LEPS** options permits a plot to be able t0 fit within these limits. For the **-EPS** option, the CALPLOT X-axis will still be horizontal. The default and **-R** option changes the lengths of the plotted axes in the manner consistent with Figures 1 and 2. The **-LEPS** option make the X-axis parallel to the long direction of the page.

The following examples use the second page of the example file *PLTTST* to illustrate the result of using this program with different options.

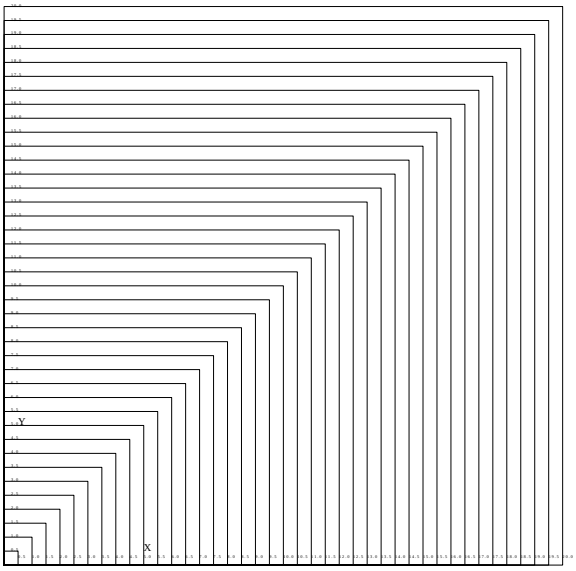


Fig. 3. This is the result of using *plotnps -EPS < plt > plttst.eps*

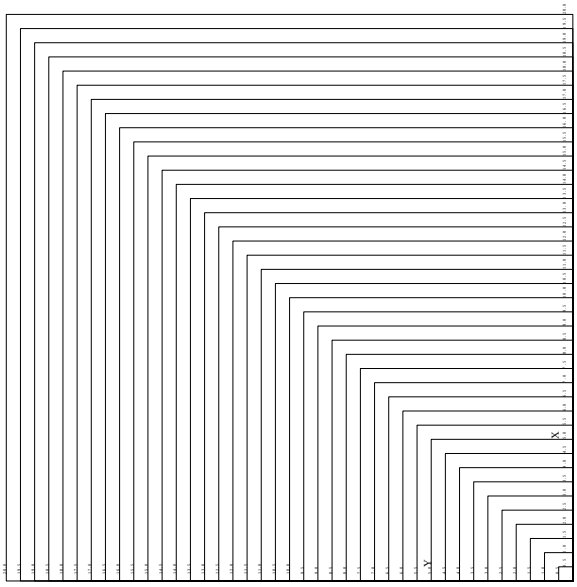


Fig. 4. This is the result of using *plotnps -EPS -R < plt > plttst.rps*

3. Windows Screen Output

The program **plotmsw** is a native WIN32 program for use under the MS Windows (WIN 95/98/NT etc) windowing system. It is built on the *graphapp* package.

Program control is through the command line:

plotmsw *[flags]*

-S*scalefac*
Scale all motions by this factor.

-F*font*
Change the default font.

-R
Rotate the plot by 90°

-N
No shading

-I
Invert the background. The background will be black instead of white. This is done by interchanging the black and white color map entries

-K
(default) Show colors with a red -> green -> blue pallette

-KR
Show colors with a red -> white -> blue pallette

-KB
Show colors with a blue -> white -> red pallette

-G
Grayscale *(color is default)*

-W*width*
Minimum linewidth in units of 0.001" or 0.0025cm

-geometry width*x*height+-xoff+-yoff
set geometry in manner of X11. The *xoff* and *yoff* are optional. *(Default width=800, height=640).*

-p
-p2
-p4
-p10
Put up background positioning grid every 1000 CALPLOT units. every 500, every 250 or every 100, respectively

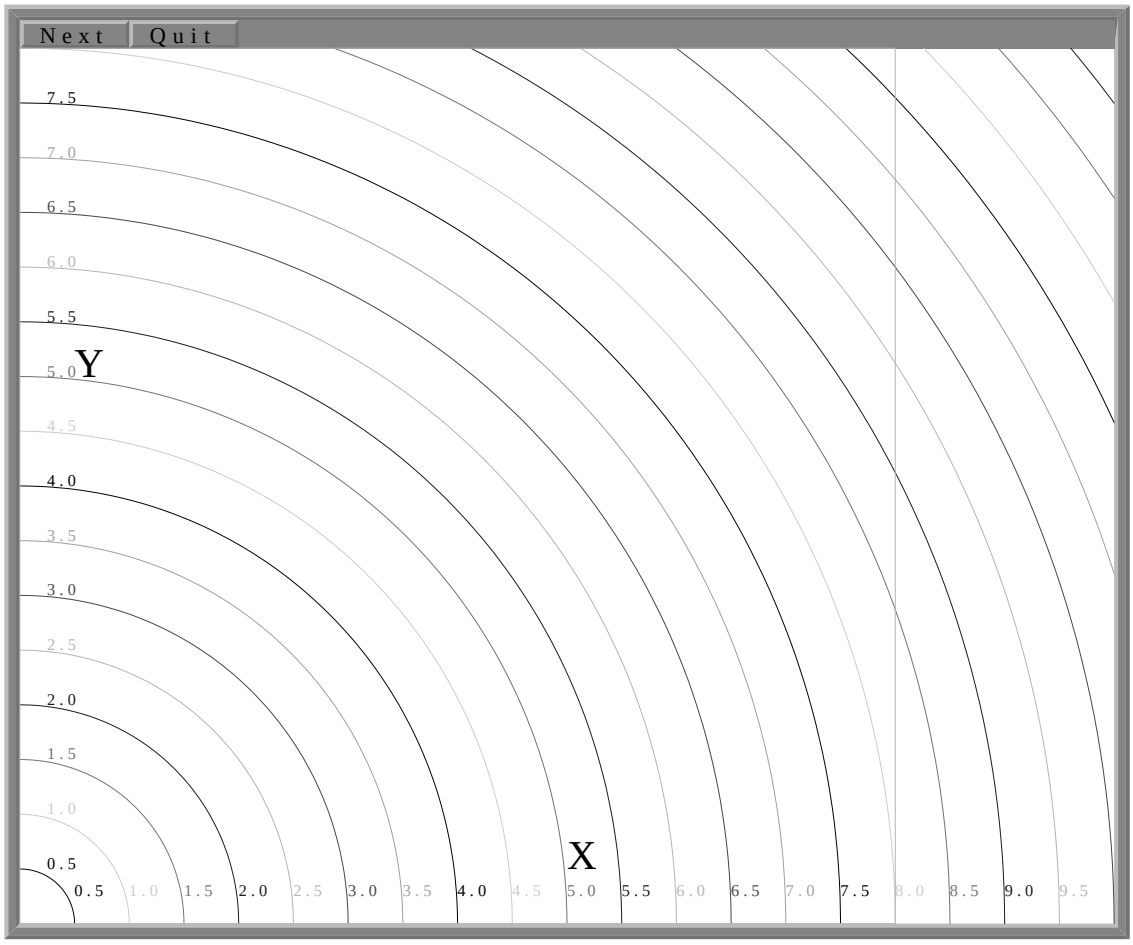
To provide additional user control, the command line arguments can be placed in the environment by separating them by colons (:) with no intervening spaces. **This is the only way to change display options when using the graphics libraries are used for interactive plots**. To force a scale factor of 0.5, and the images size of 800x600 one would set the environment parameter **PLOTMSW**

```
set PLOTMSW=:-S0.5:-g:800x600:
```

-h
-?
Online help

The Windows screen is viewed as a piece of paper exactly 10.0" wide and 8.0" high (approximately 25.4 cm wide by 20.32 cm high). The default screen has dimensions of

800 x 640 pixels.



When the page is completely drawn, a cursor will appear. One can use this to point to a feature of interest. The following actions can be performed:

- Pressing and releasing the *Left Mouse Button* will advance the page.
- Pressing the menu button *Next* will advance the page.
- Pressing the menu button *Quit* terminates the plot.

In order of importance, an entry such as this overrides a command line or environment option. For the other options, the command line overrides the PLOTMSW environment control.

Last Modified 05 NOV 2001

4. Tektronix Output

Although few will possess an actual Tektronix 4010 or 4014 graphics terminal, **kermit (3.3)** (for dial up modems) and **teraterm** (for use under Windows 3.1 and Winsock

for PPP or SLIP), support this protocol and also have added some color capability. These capabilities permit remote examination of graphics output.

Program control is through the command line:

plot4014 *[flags]* , where the command flags are

-S*scalefac*
Scale all plot motions by this factor.

-R
Rotate plot by 90°

-N
Turn off shading for speed.

-T4025
Plotting device is a Tektronix 4025 terminal.

-K
Plotting device is KERMIT 3.0. Color shading is not permitted even though color lines are. Also switch KERMIT from VT emulation to TEK emulation

-W*sleeptime*
Set a delay between plot frame. This delay is set at the computer, and will have no effect for modem or internet connections.

-D*reduc*
Reduce the resolution. The original 4014 had a 4096 x 3120 resolution. Emulator displays on PC's may have something like 500 x 400. Thus any attempt to plot greater detail will not be resolvable. If the plot driver does not send these coordinates, the result is significantly less data transmission and faster plotting. A **-D4** is useful.

-F*font*
Change the default font.

-TT
Plotting device is TERATERM. Color shading is not permitted even though color lines are is now permitted.

-h
-?
Online help

5. X11 Output

The program **plotxvig** is a native X11 program for use under the X11 windowing system. It is based on the XviG Version 1.1 package (Antoon Demarrée, IMEC, © 1993). This program supports 35 unique colors in its palette. If these colors are not available, dithering is used to create the apparent set. This package is also the basis of interactive X11 software.

Program control is through the command line:

plotxvig *[flags]*

-S*scalefac*

Scale all motions by this factor.

-F*font*
Change the default font.

-R
Rotate the plot by 90°

-N
No shading

-I
Invert the background. The background will be black instead of white. This is done by interchanging the black and white color map entries

-K
(default) Show colors with a red -> green -> blue pallette

-KR
Show colors with a red -> white -> blue pallette

-KB
Show colors with a blue -> white -> red pallette

-G
Grayscale *(color is default)*

-W*width*
Minimum linewidth in units of 0.001" or 0.0025cm

-geometry width*x*height+-xoff+-yoff
set geometry in manner of X11. The *xoff* and *yoff* are optional. *(Default width=800, height=640).*

-p
-p2
-p4
-p10
Put up background positioning grid every 1000 CALPLOT units. every 500, every 250 or every 100, respectively

To provide additional user control, the command line arguments can be placed in the environment by separating them by colons (:) with no intervening spaces. **This is the only way to change display options when using the graphics libraries are used for interactive plots**. To force a scale factor of 0.5, and the images size of 800x600 one would set the environment parameter **PLOTXVIG**

```
set PLOTXVIG=:-S0.5:-g:800x600:
export PLOTXVIG      (under sh or ksh)
```

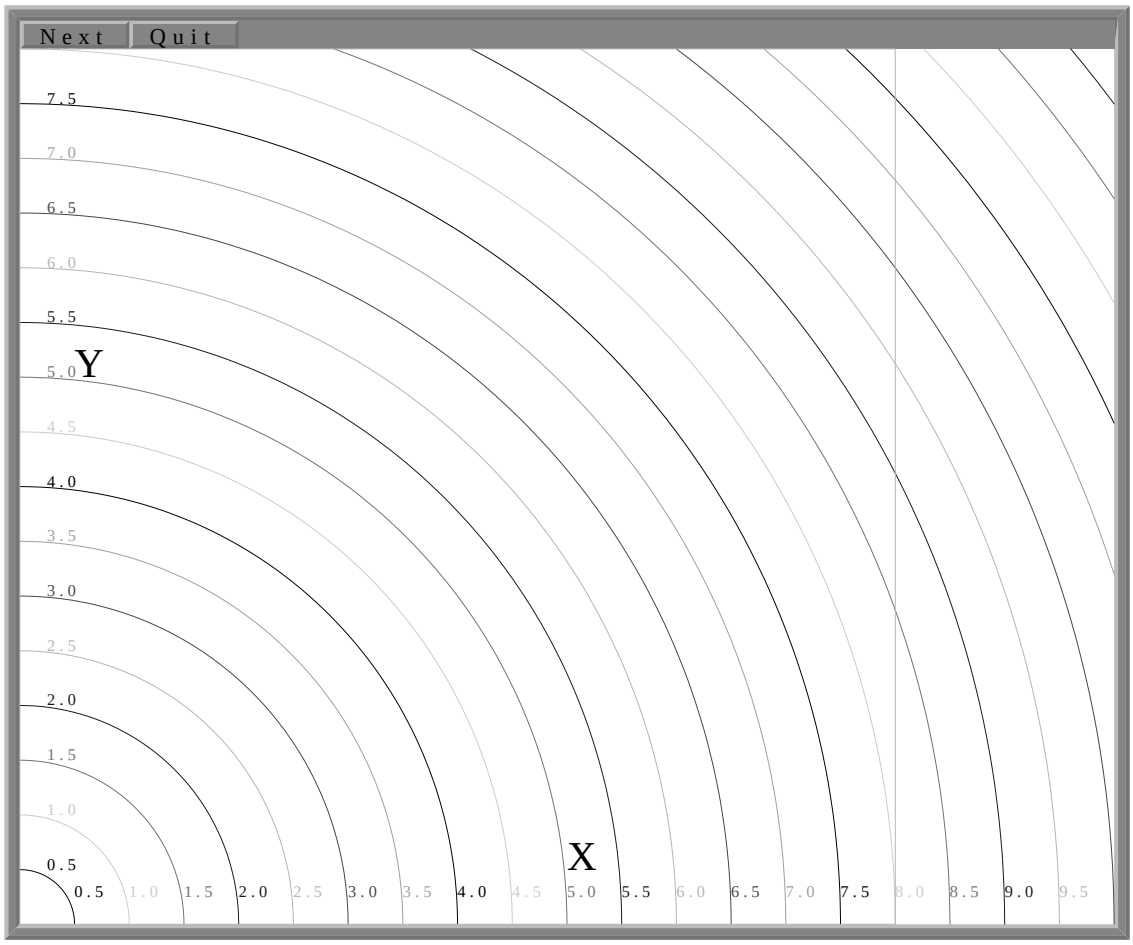
or

```
setenv PLOTXVIG=:-S0.5:-g:800x600:   (under csh)
```

-h
-?
Online help

The X11 screen is viewed as a piece of paper exactly 10.0" wide and 8.0" high (

approximately 25.4 cm wide by 20.32 cm high). The default screen has dimensions of 800 x 640 pixels, which can be changed through the window manager when the program begins. The following screen would appear:



When the page is completely drawn, a cursor will appear. One can use this to point to a feature of interest. The following actions can be performed:

- Pressing and releasing the *Left Mouse Button* will advance the page.
- Pressing the menu button *Next* will advance the page.
- Pressing the menu button *Quit* terminates the plot.

To be consistent with X11, the geometry of the plot window can be specified by an entry in the .Xdefaults file:

```
plotxvig.calxvig.plotxvig.geometry: 1000x800+100+50
```

In order of importance, an entry such as this overrides a command line or environment option. For the other options, the command line overrides the PLOTXVIG environment control.

Problems:

Since X11 programming is a new experience, here are some annoyances.

Resizing a window after plotting begins will truncate the plot, if the window is smaller, or will have unused areas. Because of the size of the binary plotfiles, there is no way to rewind and redraw a plot. Instead a backup image is used.

Finally, **`plotxvig`** works by setting up two UNIX processes: one to do do the drawing, the other to handle events and to place the drawing on the screen - an interesting use of interprocess communication. You may find yourself with a display that is not responsive - this usually happens because one, but not both processes have terminated. Use **ps** to list the processes, and then **kill `PID`** to get rid of **calxvig** and **plotxvig**.

6. PNG Output

This is a CALPLOT driver to create Portable Network Graphics (PNG) files. This CALPLOT driver was built upon the sample programs wpng.c and *writepng.c* available at *http://www.libpng.org*. This driver supports 256 unique colors in its palette which is more than the 103 colors available in CALPLOT.

Program control is through the command line:

`plotpng` *[flags] < CALPLOT_FILE > FILE.png*

`-V`
Program Version

`-S`*scalefac*
(default=1.0) Plot magnifier

`-R`
(default off) Rotate plot 90 degrees

`-N`*num*
(default 1) Convery page num

`-I`
(default off) Invert the background. The background will be black instead of white. This is done by interchanging the black and white color map entries

`-F`*font*
(default 0)
0 Roman
1 Roman
2 Italic
3 Bold
4 Symbol (Greek)

`-X`*numx*
(default 640) X-pixels one of 640,800,400,1000,2000

`-Y`*numy*
(default 480)

Y-pixels one of 480,600,320,800,1600

-K
(default color) Color output

-KW
(default -K) Color output, whitened spectrum

-KR
(default -K) Color output Red->White->Blue

-KB
(default -K) Color output Blue->White->Red

-G
(default -K) Gray output

-C*colors*
(default 256) Size of Colormap 2, 4, 16 or 256

-h
Do not execute, show options

-?
Do not execute, show options

The PNG image is viewed as a piece of paper exactly 10.0" wide and 8.0" high (approximately 25.4 cm wide by 20.32 cm high). The default image has dimensions of 640 x 480 pixels. To maintain the design aspect ratio, choose the dumension combinations 320 x 200 640 x 480, 800 x 600, 1000 x 800 or 2000 x 1600.

The program output is to *stdin* and consists of the binary PNG graphics file. It is necessary to redirect the output to a file. It is also necessary to ensure that the file name terminates with a *.png* so that other programs can recognize the file type. You can view the PNG file using a recent Web Broswer or some graphics manipulation program. An example of a command would be

```
plotpng -C16 -K -X800 -Y640 < GRAYSC.PLT > graysc.png
```

7. Figure Manipulation

The program **reframe** permits manipulation of a CALPLOT figure, either by changing the position on the page, by imposing a primitive clipping. Options exist to select one figure of a multipage plot file, and to merge plot files. The output of this program is another plot file. The program imput is from the last argument on the command line, if that argument is not a command flag, or the standard input

Program control is through the command line:

reframe *[flags]*, where the command flags are

-O
Redirect the output to the standard output. Otherwise a **plotXXXXX** file will be created, where **XXXXX** is a unique identification number.

-P
Force the output to be a plot file. This the default.

-M*mergefile*
This is the file that will be superimposed onto the original file.

-XL*x_low_clip* (default = -100000000)
-XH*x_high_clip* (default = 100000000)
-YL*y_low_clip* (default = -100000000)
-YH*y_high_clip* (default = 100000000)
A selected position of the input figure can be passed through to the output. The selected region is bounded by these coordinates.

-X0*x_origin*
-Y0*y_origin*
These values are added to the (x,y) coordinates of all input values within the clipping window to shift the resulting figure on the page.

The sequence of operations is that first the image is clipped, and then the origin is shifted.

To illustrate the usage of the program, consider the following two examples:

To merge the second frames of two plot files, one need only do

```
reframe -N2 -O -MPLOTrhwvint < PLOTrefplt > PLOTrefplt2
```

Note that the page number flag applies to both input files. It may be necessary to run the program three times to select the desired pages, first two runs, and then to merge the output using the temporary files.

To select the first three pages of a multipage plotfile, and then to combine them on to a single page,

```
reframe -V  -XH8500 -YH11000  -N1 -O < TABL > hunk1
  ( retrieve page 1 and save in the file hunk1)
reframe -V  -XH8500 -YH11000  -X0+8750  -N2 -O < TABL > hunk2
  ( retrieve page 2, move plot 8750 units to right and
     save in the file hunk2)
reframe -V  -XH8500 -YH11000  -X0+17500 -N3 -O < TABL > hunk3
  ( retrieve page 3, move plot 11000 units to right and
     save in the file hunk3)
reframe -V -N1 -O -Mhunk2 hunk1 > munk1
  ( merge the files hunk2 and hunk1 into the file munk1)
reframe -V -X0+1000 -Y0+1000 -N1 -O -Mhunk3 munk1 > PLOTreframe2
  ( merge files hunk3 and munk1, and shift the origin 1000 units to
     the right and upward)
```

All units are in the device independent plot units. When the CALPLOT programs are used, 1000 units correspond to 1.000 inches on the hardcopy plot.

The results of another example are shown in Figures 3 and 4. The object is to cut Figure 3 into four quadrants centered at (4.0,4.0) and to exchange the upper right with the lower left quadrant and the upper left with the lower right quadrant. The commands used are as follow:

```
reframe -N1 -O < PLTTST > p
reframe -N1 -O -X0+4000 -Y0+4000 -XL0000 -XH4000 -YL0000 -YH4000 < p1 > g1
reframe -N1 -O -X0-4000 -Y0+4000 -XL4000 -XH8000 -YL0000 -YH4000 < p1 > g2
reframe -N1 -O -X0+4000 -Y0-4000 -XL0000 -XH4000 -YL4000 -YH8000 < p1 > g3
reframe -N1 -O -X0-4000 -Y0-4000 -XL4000 -XH8000 -YL4000 -YH8000 < p1 > g4
reframe -N1 -O -Mg1 < g2 > g5
reframe -N1 -O -Mg3 < g4 > g6
reframe -N1 -O -Mg5 -X0+1000 -Y0+1000 < g6 > g7
plotnps -F7 -W10 -G -EPS < g7 > g7.eps
rm p1 g1 g2 g3 g4 g5 g6 g7
```

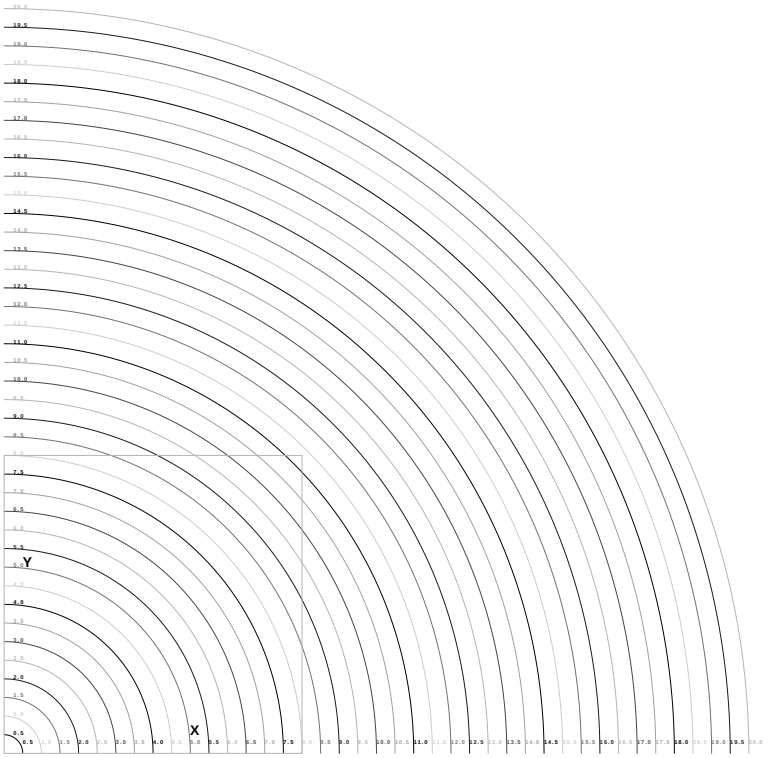


Fig. 3. Initial plot to be sectioned

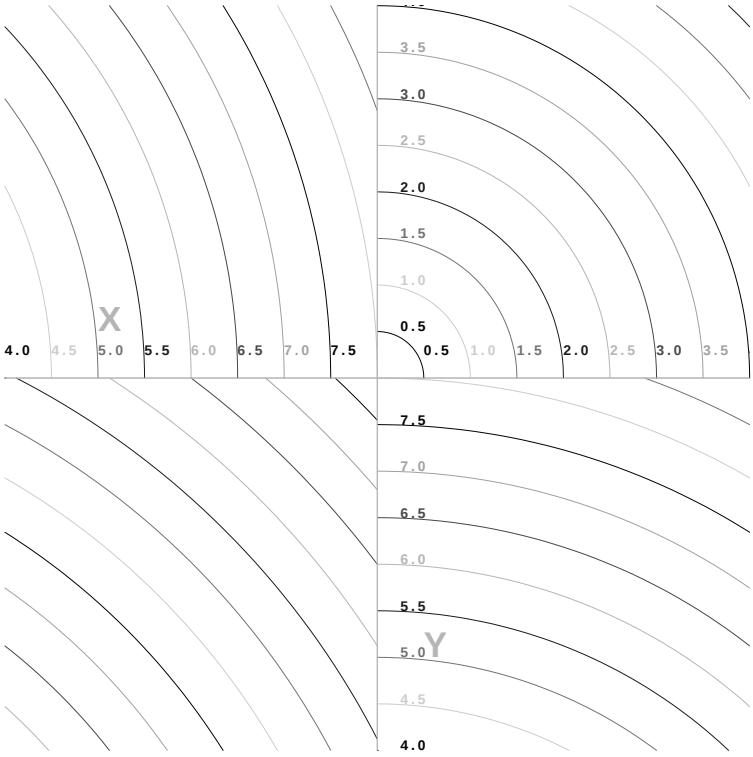


Fig. 4. Result of clipping and shifting

8. CALPLOT Colors

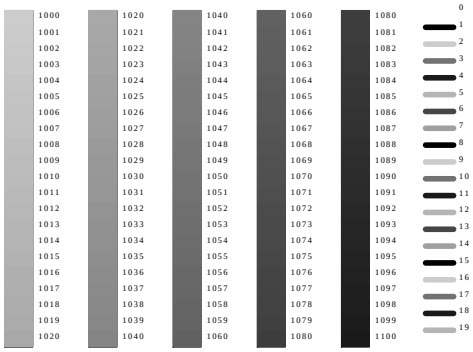
Many programs permit the user definition of colors for curves. These are invoked using

-K kolor

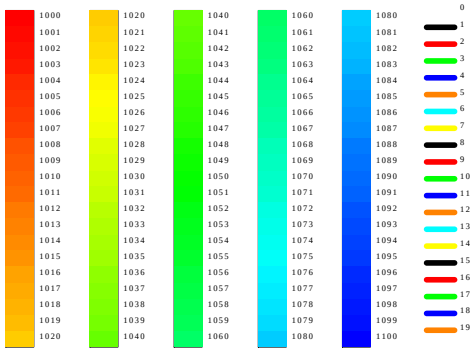
The CALPLOT graphics uses a set of predefined colors that take on slightly different meanings depending upon whether the plot program (**plotxvig, plotmsw, plot-nps, plotgif**), is invoked with the **-G**, **-K**, **-KR** or **-KB** flags. Values of kolor between **0** and **999** are mapped into a specified sequence of 7 colors. Values in the range **1000** - **1100** select a palette of continuous color tones selected by the use of these flags. The table below defines some of these values as do the figures, which are best viewed on a color terminal screen using GhostView or Acroread.

Kolor	-G	-K	-KR	-KB
0	Background	Background	Background	Background
1	Foreground	Foreground	Foreground	Foreground
2	(see below)	Red	(see below)	(see below)
3		Green		
4		Blue		
5		Orange		
6		Blue-Green		
7		Yellow		
8	Foreground	Foreground	Foreground	Foreground
9		Red		
999				
1000	Lt. Gray	Red	Red	Blue
1025		Orange	Light Red	Light Blue
1050	Med.Gray	Green	White	White
1075		BlueGreen	Light Blue	Light Red
1100	Dk. Gray	Blue	Blue	Red

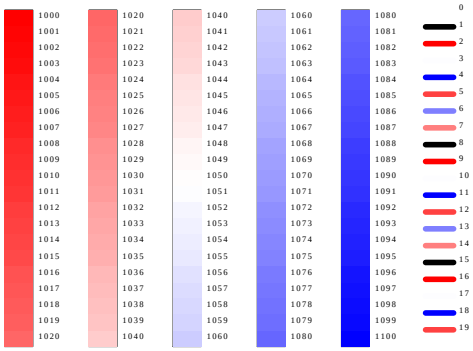
Normal plotting uses the **`-G`** and **`-K`** flags. Displays of continuous color maps can use the **`-KR`** and **`-KB`** modes if the color indices are programed to represent a range of negative - positive values with white representing a median value. The following figures show the resulting colors for a given choice of the `kolor` index.



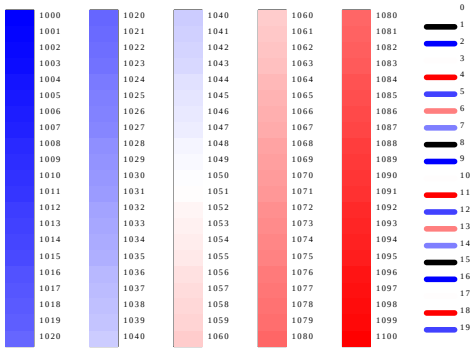
plotnps -G < GRAYSC



plotnps -K < GRAYSC



plotnps -KR < GRAYSC



plotnps -KB < GRAYSC