Contents

List of Tables

List of Figures

1 INTRODUCTION

This manual describes **TRIUMF**'s low level graphics and graph plotting software. All of the routines were developed by members of the COMPUTING SERVICES GROUP. These routines comprise a large part of the **GPLOT** library, which is a portable version of **TRIUMF**'s graphics / analysis library. The library is available on VAX/VMS, DEC ULTRIX, Silicon Graphics IRIX, ALPHA AXP OSF/1, ALPHA AXP OpenVMS, and SunOS systems. The library contains only currently used and maintained routines.

Most of the graphics software is written in FORTRAN-77, except for the Xwindow routines, which are written in C. Source code files are available for inspection and are located under `PRV41:[KOST.LIBRARY.SOURCE]` and `PRV41:[KOST.LIBRARY.SOURCE.GPLOT]`.

Routines are described in this document under the sections relevent to their function.

1.1 Low level routines

The low level graphics routines fall into two main categories:

- *control* routines that define the coordinate systems and viewing transformations
- *drawing* routines that plot points, line segments, and text

`HARDCOPY_RANGE`, `MONITOR_RANGE` and `MONITOR2_RANGE` are the fundamental control routines which define the viewing transformations. The routine `SET_PLOT_DEVICES` can be used to set up various pre-defined viewing transformations.

The fundamental drawing routines are `PLOT_R` for drawing line segments, `PLOT_POINT` for drawing points, and `PSYM` for drawing text strings. Graphics can be simultaneously generated for output to three different devices and can be directed to the graphical editor, **EDGR**, at any time. Hardcopies of the graphics can be obtained with the `GRAPHICS_HARDCOPY` routine, or through the **EDGR** program.

1.2 Graph plotting routines

The graph plotting routines are used to draw data graphs. These routines are at an intermediate level, in that they must be called as routines in user written programs, but they make use of the low level graphics package.

The fundamental graph plotting routine is `GPLOT`. Auxilliary routines `SETNAM` and `SETLAB` allow you to set values for graph and text characteristic keywords. Auxilliary routines `GETNAM` and `GETLAB` get the current values of these keywords. The `GPLOT_CONTROL` routine gives you interactive control over the graph characteristics, and also allows you to draw text strings; save and/or edit a drawing with the drawing editor; and produce graphics hardcopies.

1.3 Programs and documentation

The **EDGR** graphical editor is an interactive program for creating, editing, manipulating, and storing graphical data. Please refer to the EDGR: GRAPHICS EDITOR USER'S GUIDE for more information.

PHYSICA is a versatile, highly flexible command driven program that is capable of sophisticated data analysis as well as publication quality graphics. Please refer to the PHYSICA REFERENCE MANUAL for more information.

The TRIUMF GRAPHIC FONTS manual is a compilation of graphics font tables.

1.4 A simple program example

The following program fragment contains all of the necessary calls for plotting some data on a graph. These are some of the middle level routines that can be used for producing graphics with the **TRIUMF** software package. Each of these routines is described in detail in later chapters of this manual. These middle level routines make the calls to the low level routines for you, simplifying the process by making use of standardized coordinates, windows and viewports.

```
      ...
      CALL SET_PLOT_DEVICES(18,6,0,7,2,'IN','PORTRAIT',1)
C   select an X window monitor, plot in inches, portrait orientation
      CALL CLEAR_PLOT           ! initialize grahics and clear
      CALL GPLOTI               ! initialize graph plotting
      CALL NARGSI(4)            ! # of arguments in following call
      CALL GPLOT(X,Y,N,1)       ! graph data in X,Y arrays of length N
      CALL NARGSI(1)            ! # of arguments in following call
      CALL GRAPHICS_HARDCOPY(0) ! get a hardcopy of your picture
      ...
```

2 INSTALLATION

For best results, the installation of the **TRIUMF** software package should be undertaken by the system manager, so that all users can access its resources in a standard way.

VMS The files are typically furnished on a standard distribution VAX backup tape that also contains the **EDGR** program files and the **PHYSICA** program files. Included with the distribution is an INSTALLATION GUIDE describing the contents of the save sets and what is necessary to install these programs. The installation of the graphics library only is described here. The following files must be installed on disk:

`GPLOT.OLB`	graphics and analysis object library
`VAXFONT.DAT`	font file

The following logical definitions must be made, preferably at the system level, for access to these files:

`GPLOT$DIR`	points to the directory containing `GPLOT.OLB`
`TRIUMF$FONTS`	points to the directory containing `VAXFONT.DAT`

UNIX The files are typically furnished on a standard tar distribution tape that also contains the **EDGR** program files and the **PHYSICA** program files. The installation of the graphics library only is described here. For ease of maintenance, the following files should be kept in a common directory, for example, `/usr/users/triumf`

`gplot.a`	graphics and analysis object library
`vaxfont.dat`	font file

The following definition should be made (modified appropriately):

```
% setenv TRIUMF_FONTS /usr/users/triumf
```

The package expects to find the file `vaxfont.dat` in the directory `TRIUMF_FONTS`. If this environment variable is not defined, the package will look in `/usr/local/bin` for the file. Thus, an alternative way to install the package is to put the file in `/usr/local/bin` or to define a logical link:

```
% ln -s /usr/users/triumf/vaxfont.dat /usr/local/bin/vaxfont.dat
```

To complement this type of installation, the graphics library can also be installed in the standard place:

```
% ln -s /usr/users/triumf/libgplot.a /usr/local/lib/gplot.a
```

3 LINKING

VMS

To form an executable image, compile and link `YOURPROGRAM.FOR` code as follows:

```
$ FORTRAN YOURPROGRAM
$ LINK YOURPROGRAM,GPLOT$DIR:GPLOT/LIB
```

The logical name `GPLOT$DIR` points to the directory where the graphics library `GPLOT.OLB` is located.

UNIX

To form an executable image, compile and link `yourprogram.f` code as follows:

```
% f77 -static -c yourprogram.f
% f77 -o yourprogram yourprogram.o -lgplot -lX11
```

The library `gplot.a` must be installed as `/usr/local/lib/libgplot.a/`
Otherwise, an explicit pathname must be given for `gplot.a`

OSF1

To form an executable image, compile and link `yourprogram.f` code as follows:

```
% f77 -static -c yourprogram.f
% f77 -o yourprogram -T 00400000 -D 10000000 yourprogram.o -lgplot -lX11
```

The `-T` `-D` flags force the addresses into the two gigabyte range, which is necessary for 32 bit addressing.

The library `gplot.a` must be installed as `/usr/local/lib/libgplot.a/`
Otherwise, an explicit pathname must be given for `gplot.a`

4 SCRATCH FILES

VMS

Graphics scratch files are directed to a disk and directory via the logical name `SYS$SCRATCH`. The default value of this logical name is your login disk and directory. You can redirect the scratch file location by the following:

```
$ DEFINE SYS$SCRATCH disk:[directory]
```

UNIX

Graphics scratch files are directed to a location via the environment variable `SYS_SCRATCH`. If this environment variable is not defined, the default location for scratch file creation is given by `/tmp`. You can redirect the scratch file location with:

```
% setenv SYS_SCRATCH /your_path
```

5 GRAPHICS DEVICES

Graphics can be generated on a terminal and/or on a plotter and/or on a bitmap device. Table 5.2 on page 6 displays the currently supported graphics terminal types,[1] the plotter types,[2] and the bitmap device types. The usual mode of operation is to preview graphics on a terminal screen; and obtain hardcopy in either HP LaserJet bitmap format or in PostScript format.

terminal types	*plotter types*	*bitmap types*
DEC RETROgraphics VT640	HP 7475A/7550A	Printronix
DEC REGIS VT241	Roland GL *II*	HP LaserJet
CITOH CIT467	Houston Inst. DMP-52	HP ThinkJet
Tek 4010 / 4012	DEC LN03+	DEC LA100
Tek 4107/4109, 4207/4209	Imagen (IMPRESS)	InkJet
Plessey PT100G	GKS	
Seiko GR1105	PostScript	
X window system		
generic terminal		

Table 5.2: Graphics devices

5.1 Default terminal type

VMS The default terminal type is determined by logical name

```
$ DEFINE TRIUMF_TERMINAL_TYPE termtype
```

UNIX The default terminal type is determined by environment variable

```
% setenv TRIUMF_TERMINAL_TYPE termtype
```

The valid choices for `termtype` are displayed in Table 5.3 on page 7.

The `GET_TERMTYPE` routine returns a device name indicating the process terminal type. The name is obtained by translating `TRIUMF_TERMINAL_TYPE`. If the terminal type is undefined or cannot be obtained, the name will be blank filled.

---

[1] DEC refers to Digital Equipment Corporation, CITOH refers to CIE Terminals of Citoh Electronics, TEK refers to Tektronix Inc., HP refers to Hewlett-Packard, InkJet refers to either an HP PaintJet or to a DEC LJ250.

[2] A plotter type device is basically any graphics hardcopy device that is not a bitmap graphics device. It is not necessarily a *pen* plotter.

termtype	*description*
VT640	monochrome RETROgraphics terminal and emulators
CIT467	colour graphics CIE Terminals 467 terminal
PT100G	Plessey monochrome graphics terminal
TK4107	colour Tektronix 41xx 42xx series terminal
TK4010	monochrome Tektronix 40xx series terminal
VT241	REGIS colour graphics terminal
GR1105	Seiko colour graphics terminal
X	device running the X window system
GENERIC	terminal type defined by a user written characteristics file

Table 5.3: Choices for default terminal type

subroutine `GET_TERMTYPE(` *name* `)`

	variable	*type*	*description*
input			
output	*name*	CHARACTER*(*)	device name

Example:

```
CHARACTER*10 NAME
CALL GET_TERMTYPE(NAME)
WRITE(*,*)'NAME=',NAME
IF( NAME .EQ. 'VT640' )WRITE(*,*)'VT640 type recognized by caller'
END
```

5.2 Default plotter type

VMS The default terminal type is determined by the logical name `TRIUMF_PLOTTER_TYPE`.

```
$ DEFINE TRIUMF_PLOTTER_TYPE plottertype
```

UNIX The default terminal type is determined by the environment variable `TRIUMF_PLOTTER_TYPE`.

```
% setenv TRIUMF_PLOTTER_TYPE plottertype
```

The valid choices for `plottertype` are displayed in Table 5.4 on page 8.

The routine `GET_PLOTTERTYPE` returns a device name indicating the process plotter type. The name is obtained by translating `TRIUMF_PLOTTER_TYPE`. If the plotter type is undefined or

`plottertype`	*description*
`HPA, HPB, HPC, HPD, HPE`	different paper sizes for HP pen plotters
`HIA, HIB, HIC, HID, HIE`	different paper sizes for Houston Inst. pen plotters
`RDGLA, RDGLB, RDGLC, RDGLD, RDGLE`	different paper sizes for Roland GL pen plotters
`LN03`	DEC LN03+
`POSTSCRIPT, PSA3, PSA4`	different paper sizes for PostScript devices
`IMAGEN`	Imagen IMPRESS device
`NONE`	means no plotter file is to be made

Table 5.4: Choices for `TRIUMF_PLOTTER_TYPE`

cannot be obtained, the name will be blank filled.

subroutine `GET_PLOTTERTYPE(` *name* `)`

	variable	*type*	*description*
input			
output	*name*	`CHARACTER*(*)`	device name

Example:

```
CHARACTER*10 NAME
CALL GET_PLOTTERTYPE(NAME)
WRITE(*,*)'NAME=',NAME
IF( NAME .EQ. 'HPA' )THEN
  WRITE(*,*)'HP type recognized by caller'
  WRITE(*,*)'using size A paper'
END IF
...
```

5.3 Generic terminal

The generic terminal driver is a driver that is controlled by a data file that is created by the user. The file contains lists of escape sequences for performing the various terminal functions. Creating such a file for a given terminal monitor may allow it to be used with the graphics package even though it is not an explicitly supported device type.

VMS

To use the generic terminal driver:

```
$ DEFINE TRIUMF_TERMINAL_TYPE GENERIC
```

The name of the generic terminal file is found by translating the logical name `TRIUMF$GENTERM`.

```
$ DEFINE TRIUMF$GENTERM disk:[directory]filename.ext
```

UNIX

To use the generic terminal driver:

```
% setenv TRIUMF_TERMINAL_TYPE GENERIC
```

The name of the generic terminal file is found by translating `TRIUMF_GENTERM`.

```
% setenv TRIUMF_GENTERM /your_path/filename
```

A description of what can be in a generic terminal file is given in **Appendix A**. Example files for VT640, CIT467 and VT241 terminals, and for the KERMIT terminal emulator with VGA colour, are listed in **Appendix B**.

6 GRAPHICS HARDCOPY

Graphics hardcopy output[3] can be sent to a bitmap device queue, a plotter queue, or a file. Such a file may be in a form which can simply be printed, or it can be in a form suitable for inclusion in a TEX or LATEX document. The routine to use for obtaining graphics hardcopies is `GRAPHICS_HARDCOPY`.

subroutine `GRAPHICS_HARDCOPY(` *idevice* `)`

	variable	*type*	*description*
input	*idevice*	`INTEGER*4`	hardcopy device code
output			

The `GRAPHICS_HARDCOPY` routine requires a previous call to the `NARGSI` routine.

Example:

```
CALL NARGSI(1)
CALL GRAPHICS_HARDCOPY(0)
```

The `NARGSI` routine informs the `GRAPHICS_HARDCOPY` routine of the actual number of arguments that the `GRAPHICS_HARDCOPY` routine has been passed. This somewhat confusing arrangement is required since the `GRAPHICS_HARDCOPY` routine can be called with no arguments, that is, the *idevice* parameter defaults to zero. Under VMS, a method was developed whereby the actual number of arguments in a subroutine call could be determined at run time. The UNIX/RISC architecture uses a different argument passing mechanism from VMS. Routines with a variable number of arguments now require the number of actual arguments to be passed via a call to `NARGSI`. This does not apply to alternate return line numbers, which are optional in the argument list, but are not arguments in the usual sense.

6.1 Interactive approach

The `GRAPHICS_HARDCOPY` subroutine can be used to *interactively* obtain graphics hardcopy on any of the available devices, by passing *idevice* greater than or equal to zero.

Choosing the hardcopy device type

The *idevice* parameter can be used to pre-select the hardcopy device type. See Table 6.5 on page 11 for the numeric codes and their corresponding device types.[4]

---

[3] Hardcopies can also be obtained by directing the graphics to an **EDGR** file, and requesting the hardcopy from within the **EDGR** program.

[4] The GKS metafile is available only at sites which have a local GKS library. In this case, there will likely be an interpreter program which allows the metafile to be replayed onto various printers and terminals.

idevice	*device type*
0	device not specified
1	Printronix
3	HP pen plotter
4	HP LaserJet
5	HP ThinkJet
6	DEC LA100
8	Houston Instruments pen plotter
9	DEC LN03+
10	IMAGEN
11	InkJet
12	PostScript
14	GKS metafile
15	Roland GL pen plotter

Table 6.5: `GRAPHICS_HARDCOPY` numeric device codes

If *idevice* is zero, the bitmap and plotter settings are tested and a menu of possible devices is displayed, see Table 6.6 on page 12. The user is expected to enter a character code to select the hardcopy device type.

Note: This first code request is bypassed if only one type of output device is enabled, or if *idevice* is non-zero in the call to `GRAPHICS_HARDCOPY`.

How the bitmap device is determined

The bitmap device type is determined by decoding the variable `IBIT` in the `BITMAP_DEVICE` common block.

```
INTEGER*4 IBIT
COMMON \BITMAP_DEVICE\ IBIT
```

Table 6.7 on page 12 shows the bitmap device codes.

Hardcopy output options

Following the device code, if applicable, the user is requested to enter a hardcopy command code for printing or saving the graphics for the chosen device type. This code is a one or two letter code usually followed by a queue name or a file name. Refer to Table 6.8 on page 13. Command code parameters enclosed in curly brackets, `{ }`, assume default values when

code	*device type*
`P`	Printronix
`HPP`	HP pen plotter
`HPL`	HP LaserJet
`HPT`	HP ThinkJet
`LA`	DEC LA100
`HOU`	Houston Instruments pen plotter
`LN`	LN03+
`IM`	IMAGEN
`HPJ`	InkJet
`PS`	PostScript
`GKS`	GKS metafile
`RDG`	Roland GL pen plotter

Table 6.6: `GRAPHICS_HARDCOPY` character device codes

`IBIT`	*device type*
0	no bitmap
1	Printronix
2	HP LaserJet 100 dpi
12	HP LaserJet 150 dpi
32	HP LaserJet 300 dpi
3	HP ThinkJet
4	DEC LA100
5	InkJet

Table 6.7: `IBIT` variable bitmap device codes

not entered.

device name	*device code*	*command code*	*parameter*	*action*	*default*
Printronix	P	P	queue	print	Lnptr, Trmf, or other
		S	{ file }	save file	PX.PLT
HP plotter	HPP	P	{ queue }	print	HPLTR
		S	{ file }	save file	HPP.PLT
		A		auxiliary port output	
HP LaserJet	HPL	P	{ queue }	print	HP$LASER
		PC	{ queue }	print in compressed format	HP$LASER
		S	{ file }	save file	HPLASER.PLT
		SC	{ file }	save file in compressed format	HPLASER.PLT
		A		auxiliary port output	
		T	{ file }	TeX output file	HPTEX.PLT
		TC	{ file }	TeX output file compressed	HPTEX.PLT
		TJ	{ file }	TeX justified output	HPTEX.PLT
		TJC	{ file }	TeX justified output compressed	HPTEX.PLT
HP ThinkJet	HPT	P	queue	print	
		S	{ file }	save file	HPTHINK.PLT
		A		auxiliary port output	
DEC LA100	LA	P	{ queue }	print	PHYS
		S	{ file }	save file	LA100.PLT
		A		auxiliary port output	
Houston Instruments	HOU	P	queue	print	
		S	{ file }	save file	HOUSTON.PLT
		A		auxiliary port output	
LN03+	LN	P	queue	print	
		S	{ file }	save file	LN03.PLT
IMAGEN	IM	P	queue	print	
		S	{ file }	save file	IMAGEN.PLT
InkJet	HPJ	P	queue	print	
		PT	{ queue }	print transparency	
		S	{ file }	save file	HPPAINT.PLT
		A		auxiliary port output	
PostScript	PS	P	{ queue }	print	POST$SCRIPT
		S	{ file }	save file	POSTSCRIPT.PLT
GKS metafile	GKS	S	{ file }	save file	GKSMETA.PLT
Roland GL	RDG	P	{ queue }	print	RDGL
		S	{ file }	save file	RDGL.PLT
		A		auxiliary port output	

Table 6.8: `GRAPHICS_HARDCOPY` command codes

Note: Compressed format can speed up the printing of a drawing by as much as a factor of 5. Compressed format is recognized *only* by the Hewlett-Packard LaserJet IIP and later printers. Do **not** attempt to print compressed format output on other than these devices.

VMS | In the print operation of the `GRAPHICS_HARDCOPY` routine, intermediate plot files are written to the disk and directory specified by the logical name `SYS$SCRATCH`.

UNIX | In the print operation of the `GRAPHICS_HARDCOPY` routine, intermediate plot files are written to the `/tmp` directory.

6.2 Non-interactive approach

The routine `GRAPHICS_HARDCOPY` can be used to *non-interactively* obtain graphics hardcopy, by passing $idevice$ less than zero.

If $idevice$ is less than zero, the variable `QUE_NAME` is used to reference the name of the queue on which to automatically print the plot file. The variable `QUE_NAME` is passed in the `QUE_NAMES` common block.

```
CHARACTER*20 QUE_NAME
COMMON /QUE_NAMES/ QUE_NAME
```

Example:

```
      CHARACTER*20 QUE_NAME
      COMMON /QUE_NAMES/ QUE_NAME
      QUE_NAME = 'LASER_119'
      ...
C   call low level graphics control routines
      CALL CLEAR_PLOT    ! to initialize the drawing routines
C   call graph plotting routines and/or drawing routines
      ...
      CALL NARGSI(1)
      CALL GRAPHICS_HARDCOPY(-4)
      END
```

The numeric code for the device type will be $|idevice|$. See Table 6.5 on page 11 for a listing of the numeric device codes and their corresponding device types.

6.3 Printing a plot file on a device queue

When printing a plot file that has been previously saved, be certain that the appropriate device queue is chosen, that is, a device queue corresponding to the type of plot file that was saved.

VMS

To print a bitmap plot file, issue the DCL command:

```
$ PRINT/PASSALL/QUEUE=queuename plotfile
```

To print a plotter file, issue the DCL command:

```
$ PRINT/NOHEAD/NOFEED/QUEUE=queuename plotfile
```

UNIX

To print a bitmap plot file, issue the command:

```
% lpr -x -Pprintername plotfile
```

To print a plotter file, issue the command:

```
% lpr -h -Pprintername plotfile
```

6.4 Inserting a plot into a document

Graphics generated by the **TRIUMF** software can be included as figures into a TEX or LATEX document. **TRIUMF** has device interface programs for HP LaserJet printers and for PostScript.

To include a figure for HP LaserJet output using the DVIHP program, make sure you select a TEX output option when the hardcopy plot file is made. Refer to Table 6.8 on page 13. The plot file must be specifically made for TEX inclusion. Regular `HPLASER` plot files will not work!

To include a figure for PostScript using the DVIPS program, simply request a PostScript hardcopy output file. Again, refer to Table 6.8 on page 13.

For information on using the programs which convert device independent TEX files into printable files, and on how to insert plots into your documents, please refer to the TEX AND LATEX AT TRIUMF manual.

7 COORDINATE SYSTEMS

The **TRIUMF** graphics system employs four **coordinate systems** that are linked by **viewing transformations**. All coordinate systems are two dimensional Cartesian.[5] Graphical objects are first defined in **world coordinates**, which is any system of units that is convenient for the user's particular application. World coordinates are mapped into **bitmap coordinates**, which are the pixel coordinates of the bitmap. Bitmap coordinates are mapped into **monitor coordinates**, which are the physical device coordinates of the terminal and/or plotter. Since two different monitors can be simultaneously active, there are two, independent, monitor coordinate systems.

A two-stage viewing transformation is used to generate images on the viewing surface(s) from world coordinates. The first stage maps a rectangular region in world coordinate space, called the **world window**, to a rectangular region in bitmap space, called the **bitmap viewport**. The second stage maps a rectangular region in bitmap coordinate space, called the **bitmap window**, to a rectangular region in monitor coordinate space, called the **monitor viewport**. If two monitors are active, there will be seperate viewing transformations from bitmap space to the first monitor space and from bitmap space to the second monitor space. Figure 7.2 shows the most general transformations linking the world system, the bitmap system, and the monitor systems, while Figure 7.3 shows the commonly used transformations linking the windows and the viewports.

The windows and viewports are defined by the low level routines:

`HARDCOPY_RANGE`, `MONITOR_RANGE`, and `MONITOR2_RANGE`.

`HARDCOPY_RANGE` *must* be called before `MONITOR_RANGE` and `MONITOR2_RANGE`. These routines need not be called if the default transformations are acceptable, or if some set-up routine, such as `SET_PLOT_DEVICES`, is called to select from a set of pre-defined transformations.

Different bitmap windows and monitor viewports can be specified for the two monitors, allowing different portions of a picture to be seperately generated on the monitors, see Figure 7.2 on page 17. Also, the viewing transformations can be changed during a graphics session by re-invoking `HARDCOPY_RANGE`, `MONITOR_RANGE` and/or `MONITOR2_RANGE`.

7.1 World coordinates

World coordinates, denoted here by (x_w, y_w), can employ whatever units are most convenient for the user's application. The default units are $0 \leq x_w \leq 639$ and $0 \leq y_w \leq 479$, but any

---

[5] Although higher-level graphics packages may use 3-dimensional, polar, or other specialized coordinates, they will ultimately draw lines in Cartesian coordinates.

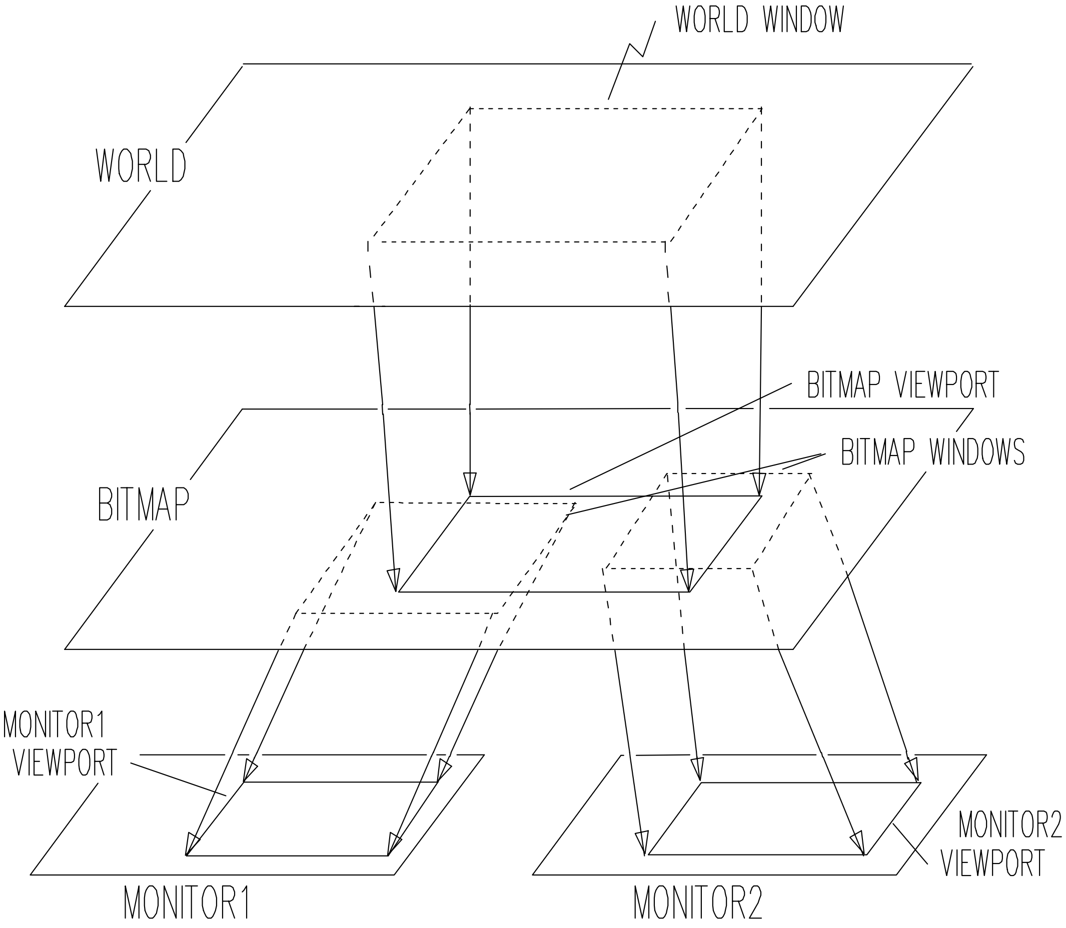


Figure 7.2: Possible viewing transformations

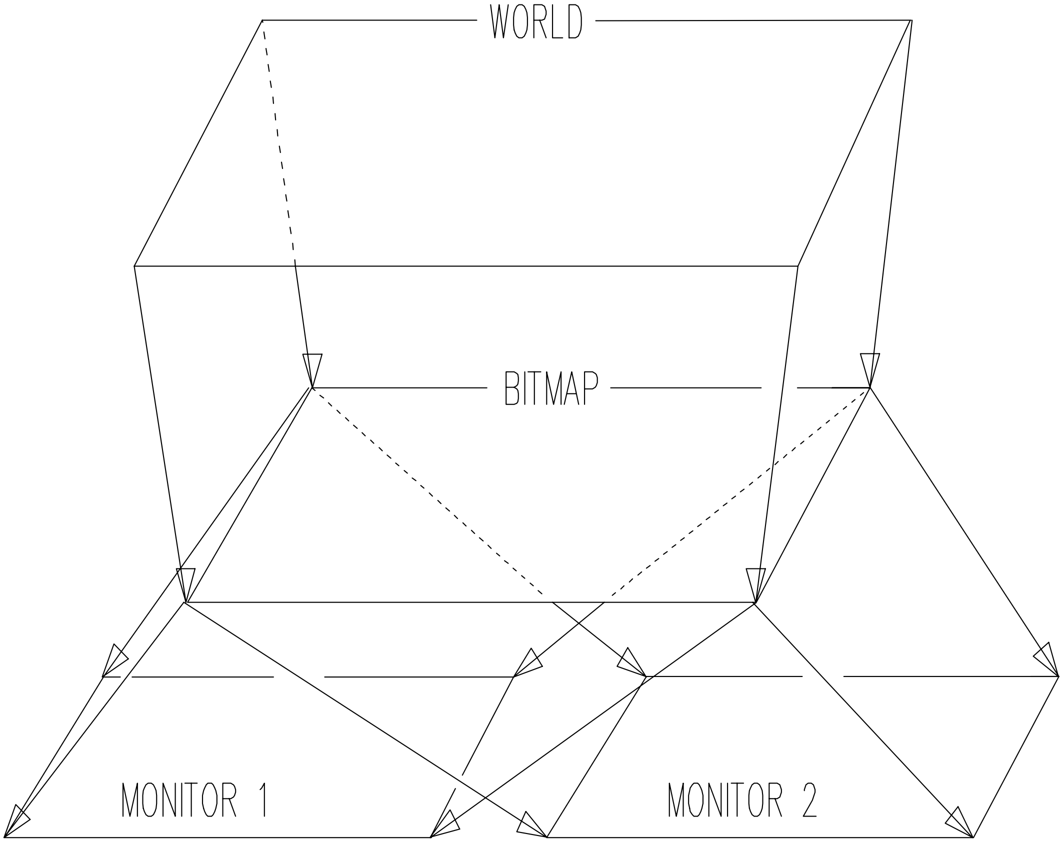


Figure 7.3: Commonly used viewing transformations

coordinate system is allowed. Typical user defined world coordinate systems are $0 \leq x_w \leq 1$ and $0 \leq y_w \leq 1$, or $0 \leq x_w \leq 11$ and $0 \leq y_w \leq 8.5$

7.2 Bitmap coordinates

Bitmap coordinates, denoted here by (x_b, y_b), serve as an intermediate coordinate system for mapping the world coordinates to the monitor window(s). A bitmap is maintained, which can be copied to a bitmap device. This bitmap is a quantized image of the bitmap coordinate system. The full resolution of the graphics drawing is maintained during the transformation from world to bitmap coordinates, that is, the quantization of the bitmap does not affect the quality of the monitor images.

Bitmap coordinates are restricted to be within various limits, dependent on the chosen bitmap device. The default bitmap units are $0 \leq x_b \leq 479$ and $0 \leq y_b \leq 639$.

7.3 Monitor coordinates

Monitor coordinates, denoted here by (x_m, y_m), are restricted to the physical coordinates of the devices. If two monitors are activated, the two monitors can display the same or different portions of the world coordinate space. Usually, the first monitor is a terminal and the second monitor is a plotter. Seventeen types of monitor are currently supported. Table 5.2 on page 6 shows the terminal types and the plotter types. For each monitor type, Table 7.9 on page 20 shows the limits for (x_m, y_m) as well as the monitor type codes.

For terminal monitors, the plotting routines automatically quantize the generated (x_m, y_m) pairs to map to the respective pixel ranges of the terminals. On these terminal monitors, the x_m axis is horizontal and y_m is vertical, with the origin in the lower left hand corner.

For the pen plotter type monitors: HP (type code 5), Houston Instruments (type code 11), and Roland GL (type code 20), the (x_m, y_m) pairs map to distance in centimetres on the paper surface, where the x_m axis is horizontal and the y_m axis is vertical.

Table 7.10 on page 20 shows the standard paper sizes allowed for pen plotters and some PostScript printers. Not all devices accept all these sizes of paper. The user must check this beforehand and, sometimes, manually load the correct size paper. The plotting region on the paper is usually made smaller than the maximum physical paper area.

The default monitor units are VT640 terminal screen pixels, $0 \leq x_m \leq 639$ and $0 \leq y_m \leq 479$. There is no default second monitor.

	type code	x_m *min*	x_m *max*	y_m *min*	y_m *max*
no graphics monitor	0	–	–	–	–
DEC RETROgraphics VT640	1	0	639	0	479
Tek 4010 / 4012	2	0	1023	0	779
HP pen plotter	5	0	109.22	0	83.82
CITOH CIT467	6	0	571	0	479
Tek 4107/4109, 4207/4209	7	0	639	0	479
REGIS VT241	8	0	639	0	479
Plessey PT100G	9	0	639	0	479
Houston Inst. DMP-52	11	0	109.22	0	83.82
Seiko GR1105	12	−2048	2047	−1560	1559
Imagen (IMPRESS)	13	0	3328	0	2560
PostScript	14	0	3357	0	2400
DEC LN03+	16	0	639	0	479
generic terminal	17				
X window system	18	0	1	0	0.75
GKS metafile	19	0	1	0	1
Roland GL pen plotter	20	0	21272	0	16672

Table 7.9: Monitor type codes and coordinate limits

	centimeters		*inches*	
size code	x *max*	y *max*	x *max*	y *max*
`A`	27.94	21.59	11.00	8.50
`B`	40.64	25.40	17.00	11.00
`C`	53.34	40.64	22.00	17.00
`D`	83.82	53.34	34.00	22.00
`E`	109.22	83.82	44.00	34.00
`A3`	27.94	21.59	11.00	8.50
`A4`	29.70	21.00	11.69	8.27

Table 7.10: Standard paper sizes

8 FUNDAMENTAL CONTROL ROUTINES

The fundamental control routines define the viewing transformations. They are `HARDCOPY_RANGE`, which defines the world window and the bitmap viewport, `MONITOR_RANGE`, which defines a window on the bitmap viewport and the first monitor viewport, and `MONITOR2_RANGE`, which defines another window on the bitmap viewport and the second monitor viewport.

The `SET_PLOT_DEVICES` routine calls the fundamental control routines and can be used to set up various pre-defined viewing transformations.

8.1 HARDCOPY_RANGE

The `HARDCOPY_RANGE` routine defines the world window and the bitmap viewport.

subroutine `HARDCOPY_RANGE(` $x_{B\downarrow}$, $x_{B\uparrow}$, $y_{B\downarrow}$, $y_{B\uparrow}$, $x_{w\downarrow}$, $x_{w\uparrow}$, $y_{w\downarrow}$, $y_{w\uparrow}$, ior_b `)`

	variable	*type*	*description*
input	$x_{B\downarrow}$	REAL*4	bitmap viewport x minimum
	$x_{B\uparrow}$	REAL*4	bitmap viewport x maximum
	$y_{B\downarrow}$	REAL*4	bitmap viewport y mimimum
	$y_{B\uparrow}$	REAL*4	bitmap viewport y maximum
	$x_{w\downarrow}$	REAL*4	world window x minimum
	$x_{w\uparrow}$	REAL*4	world window x maximum
	$y_{w\downarrow}$	REAL*4	world window y mimimum
	$y_{w\uparrow}$	REAL*4	world window y maximum
	ior_b	INTEGER*4	bitmap orientation
output			

The limits of the bitmap viewport, in bitmap coordinate units are defined by $x_{B\downarrow}$, $x_{B\uparrow}$, $y_{B\downarrow}$, and $y_{B\uparrow}$. Similarly, $x_{w\downarrow}$, $x_{w\uparrow}$, $y_{w\downarrow}$, and $y_{w\uparrow}$ define the limits of the **world window**, in the user defined world coordinate system.

The orientation of the bitmap is defined by ior_b, which controls the presence or absence of a rotation with respect to the bitmap coordinate axes when mapping from world to bitmap coordinates. If $ior_b = +1$ then there is no rotation, while if $ior_b = -1$ then there is a $90°$ counterclockwise rotation. See the **Orientation** section for more information.

The world window, as defined in the call to `HARDCOPY_RANGE`, can be found in the `HARDCOPYRANGE` common block:

```
REAL*4    XMINW, XMAXW, YMINW, YMAXW
INTEGER*4 IORW
COMMON /HARDCOPYRANGE/ XMINW, XMAXW, YMINW, YMAXW, IORB
```

where XMINW $= x_{w\downarrow}$, XMAXW $= x_{w\uparrow}$, YMINW $= y_{w\downarrow}$, YMAXW $= y_{w\uparrow}$, and IORB $= ior_b$

The bitmap viewport, as defined in the call to HARDCOPY_RANGE, can be found in the HARDCOPYRANGE2 common block:

```
REAL*4 XMINB, XMAXB, YMINB, YMAXB
COMMON /HARDCOPYRANGE2/ XMINB, XMAXB, YMINB, YMAXB
```

where XMINB $= x_{B\downarrow}$, XMAXB $= x_{B\uparrow}$, YMINB $= y_{B\downarrow}$, and YMAXB $= y_{B\uparrow}$.

8.2 MONITOR_RANGE

The MONITOR_RANGE routine defines a window on the bitmap viewport and the first monitor viewport.

subroutine MONITOR_RANGE(t_1, u_1, $x_{b\downarrow}$, $x_{b\uparrow}$, $y_{b\downarrow}$, $y_{b\uparrow}$, $x_{m\downarrow}$, $x_{m\uparrow}$, $y_{m\downarrow}$, $y_{m\uparrow}$, ior_m)

	variable	*type*	*description*
input	t_1	INTEGER*4	monitor 1 type code
	u_1	INTEGER*4	monitor 1 logical unit number
	$x_{b\downarrow}$	REAL*4	bitmap viewport x minimum
	$x_{b\uparrow}$	REAL*4	bitmap viewport x maximum
	$y_{b\downarrow}$	REAL*4	bitmap viewport y mimimum
	$y_{b\uparrow}$	REAL*4	bitmap viewport y maximum
	$x_{m\downarrow}$	REAL*4	monitor 1 window x minimum
	$x_{m\uparrow}$	REAL*4	monitor 1 window x maximum
	$y_{m\downarrow}$	REAL*4	monitor 1 window y mimimum
	$y_{m\uparrow}$	REAL*4	monitor 1 window y maximum
	ior_m	INTEGER*4	monitor 1 orientation
output			

The monitor type code is t_1. See Table 7.9 on page 20. For example, $t_1 = 1$ is a VT640 terminal, while $t_1 = 5$ is an HP pen plotter. The logical unit number for output to the first monitor is the u_1 parameter.

The parameters $x_{b\downarrow}$, $x_{b\uparrow}$, $y_{b\downarrow}$, and $y_{b\uparrow}$ define the first monitor's window on the bitmap viewport. Usually this window is the entire viewport, that is, $x_{b\downarrow} = x_{B\downarrow}$, $x_{b\uparrow} = x_{B\uparrow}$, $y_{b\downarrow} = y_{B\downarrow}$, and $y_{b\uparrow} = y_{B\uparrow}$.

The parameters $x_{m\downarrow}$, $x_{m\uparrow}$, $y_{m\downarrow}$, and $y_{m\uparrow}$ define the first monitor viewport, which should be expressed in the physical units of the first monitor device.

The first monitor orientation is defined by ior_m, which controls the presence or absence of a rotation with respect to the first monitor coordinate axes, when mapping from bitmap to first monitor coordinates. If $ior_m = +1$ then there is no rotation, while if $ior_m = -1$ then there is a 90° clockwise rotation. See the **Orientation** section for more information.

The monitor viewport, as defined in the call to `MONITOR_RANGE`, as well as these boundary values transformed into world coordinates, can be found in the `MONITORRANGE` common block:

```
      REAL*4 XMINMW,XMAXMW,YMINMW,YMAXMW,XMINM,XMAXM,YMINM,YMAXM
      COMMON /MONITORRANGE/ XMINMW, XMAXMW, YMINMW, YMAXMW,
     #                      XMINM,  XMAXM,  YMINM,  YMAXM
```

where `XMINM` $= x_{m\downarrow}$, `XMAXM` $= x_{m\uparrow}$, `YMINM` $= y_{m\downarrow}$, and `YMAXM` $= y_{m\uparrow}$, and where `XMINMW`, `YMINMW`, `XMAXMW`, and `YMAXMW` are the transformed values.

The monitor type and the output unit number for the first monitor can be found in the `PLOTMONITOR` common block.

```
      INTEGER*4 IMONITOR, IOUTM
      COMMON /PLOTMONITOR/ IMONITOR, IOUTM
```

where `IMONITOR` $= t_1$ and `IOUTM` $= u_1$, as defined in the call to `MONITOR_RANGE`.

8.3 MONITOR2_RANGE

`MONITOR2_RANGE` defines another window on the bitmap viewport and the second monitor viewport. Usually, the second monitor is a plotter.

subroutine `MONITOR2_RANGE(` t_2, u_2, $x_{b\downarrow}$, $x_{b\uparrow}$, $y_{b\downarrow}$, $y_{b\uparrow}$, $x_{m\downarrow}$, $x_{m\uparrow}$, $y_{m\downarrow}$, $y_{m\uparrow}$, ior_m `)`

	variable	*type*	*description*
input	t_2	`INTEGER*4`	monitor 2 type code
	u_2	`INTEGER*4`	monitor 2 logical unit number
	$x_{b\downarrow}$	`REAL*4`	bitmap viewport x minimum
	$x_{b\uparrow}$	`REAL*4`	bitmap viewport x maximum
	$y_{b\downarrow}$	`REAL*4`	bitmap viewport y mimimum
	$y_{b\uparrow}$	`REAL*4`	bitmap viewport y maximum
	$x_{m\downarrow}$	`REAL*4`	monitor 2 window x minimum
	$x_{m\uparrow}$	`REAL*4`	monitor 2 window x maximum
	$y_{m\downarrow}$	`REAL*4`	monitor 2 window y mimimum
	$y_{m\uparrow}$	`REAL*4`	monitor 2 window y maximum
	ior_m	`INTEGER*4`	monitor 2 orientation
output			

The monitor type code is t_2. See Table 7.9 on page 20. For example, $t_2 = 1$ is a VT640 terminal, while $t_2 = 5$ is an HP pen plotter. The logical unit number for output to the second monitor is the u_2 parameter.

VMS

If the second monitor is a plotter, output to u_2 is directed to a scratch file located on `SYS$SCRATCH`.

UNIX

If the second monitor is a plotter, output to u_2 is directed to a scratch file located on `/tmp`.

The parameters $x_{b\downarrow}$, $x_{b\uparrow}$, $y_{b\downarrow}$, and $y_{b\uparrow}$ define the second monitor's window on the bitmap viewport. Usually this window is the entire viewport, that is, $x_{b\downarrow} = x_{B\downarrow}$, $x_{b\uparrow} = x_{B\uparrow}$, $y_{b\downarrow} = y_{B\downarrow}$, and $y_{b\uparrow} = y_{B\uparrow}$.

The parameters $x_{m\downarrow}$, $x_{m\uparrow}$, $y_{m\downarrow}$, and $y_{m\uparrow}$ define the second monitor viewport, which should be expressed in the physical units of the second monitor device.

The second monitor orientation is defined by ior_m, which controls the presence or absence of a rotation with respect to the second monitor coordinate axes, when mapping from bitmap to second monitor coordinates. If $ior_m = +1$ then there is no rotation, while if $ior_m = -1$ then there is a $90°$ clockwise rotation. See the **Orientation** section for more information.

The monitor viewport, as defined in the call to `MONITOR2_RANGE`, as well as these boundary values transformed into world coordinates, can be found in the `MONITOR2RANGE` common block:

```
      REAL*4 XMINMW,XMAXMW,YMINMW,YMAXMW,XMINM,XMAXM,YMINM,YMAXM
      COMMON /MONITOR2RANGE/ XMINMW, XMAXMW, YMINMW, YMAXMW,
     #                       XMINM,  XMAXM,  YMINM,  YMAXM
```

where `XMINM` $= x_{m\downarrow}$, `XMAXM` $= x_{m\uparrow}$, `YMINM` $= y_{m\downarrow}$, and `YMAXM` $= y_{m\uparrow}$, and where `XMINMW`, `YMINMW`, `XMAXMW`, and `YMAXMW` are the transformed values.

The monitor type and the output unit number for the second monitor can be found in the `PLOTMONITOR2` common block.

```
      INTEGER*4 IMONITOR2, IOUTM2
      COMMON /PLOTMONITOR2/ IMONITOR2, IOUTM2
```

where `IMONITOR2` $= t_2$ and `IOUTM2` $= u_2$, as defined in the call to `MONITOR2_RANGE`.

8.4 Default viewing transformations

In the absence of calls to `HARDCOPY_RANGE`, `MONITOR_RANGE`, `MONITOR2_RANGE`, or some other setup routine which calls these subroutines, the default windows, viewports, orientations and monitor type are shown in Table 8.11 on page 25. See also Example 1 in the **Examples of Windows and Viewports** section.

	x min	*x max*	*y min*	*y max*	*orientation*
world	0	639	0	479	
bitmap	0	479	0	639	-1
monitor	0	639	0	479	-1

Table 8.11: Default viewing transformations

The default monitor type is 1, a VT640 terminal type, writing on logical unit number 6.

Note: There is no default second monitor.

8.5 Clipping

Objects are clipped at the boundary of the world window, meaning that lines and points outside of the **world window** are not mapped to the bitmap viewport.

Objects in the bitmap viewport are seperately clipped at each bitmap window before they are mapped to the respective monitor viewports.

8.6 Orientation

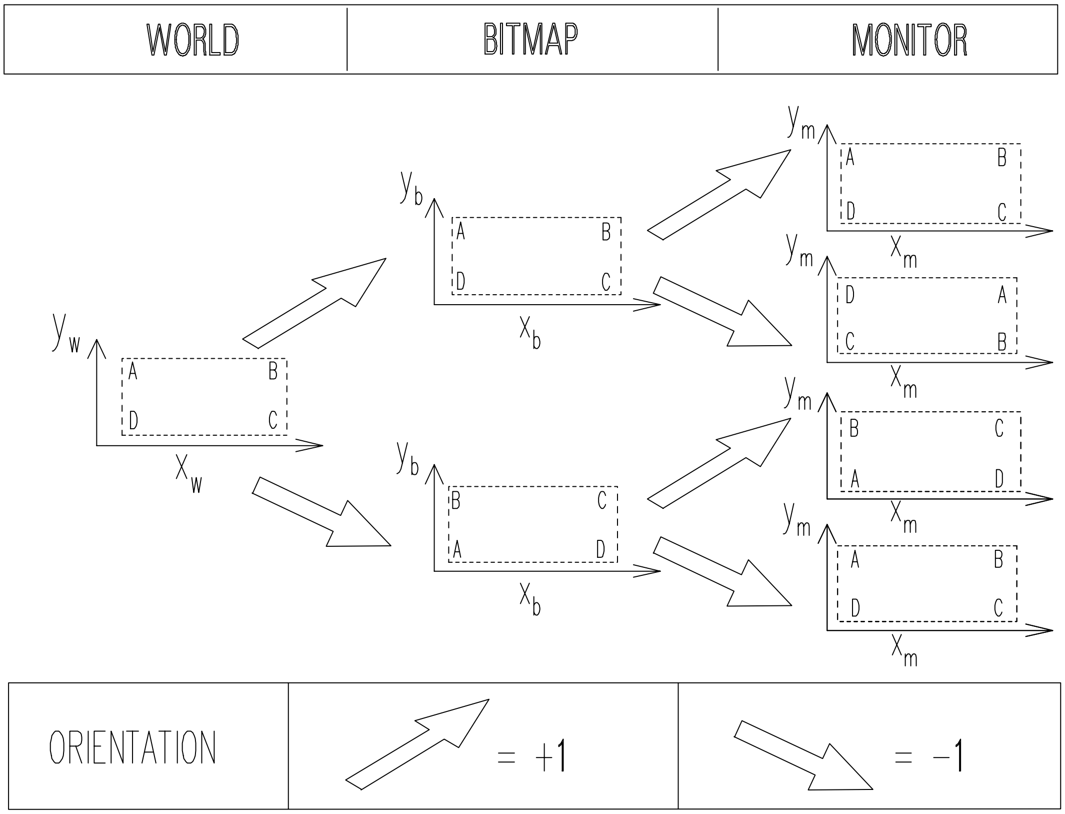


Figure 8.4: Effects of orientation parameters

The viewing transformations can involve a change in orientation, that is, a $90°$ rotation of the image, when mapping from world to bitmap coordinates and from bitmap to monitor coordinates.

The bitmap orientation parameter, ior_b, as defined in the call to `HARDCOPY_RANGE`, controls the presence or absence of a $90°$ counterclockwise rotation with respect to the bitmap coordinate axes when mapping from world to bitmap coordinates. The monitor orientation parameter, ior_m, as defined in the call to `MONITOR_RANGE` or `MONITOR2_RANGE`, controls the presence or absence of a $90°$ clockwise rotation with respect to the monitor coordinate axes when mapping from bitmap to monitor coordinates. See Figure 8.4 on page 26 and Table 8.12 on page 27.

	result
$ior_b = -1$	Rotate the image 90° counterclockwise. A point at $(x_{w\downarrow}, y_{w\downarrow})$ is mapped to $(x_{B\uparrow}, y_{B\downarrow})$ and $(x_{w\uparrow}, y_{w\uparrow})$ is mapped to $(x_{B\downarrow}, y_{B\uparrow})$.
$ior_b = +1$	Do not rotate. A point at $(x_{w\downarrow}, y_{w\downarrow})$ is mapped to $(x_{B\downarrow}, y_{B\downarrow})$ and $(x_{w\uparrow}, y_{w\uparrow})$ is mapped to $(x_{B\uparrow}, y_{B\uparrow})$.
$ior_m = -1$	Rotate the image 90° clockwise. A point at $(x_{b\downarrow}, y_{b\downarrow})$ is mapped to $(x_{m\downarrow}, y_{m\uparrow})$ and $(x_{b\uparrow}, y_{b\uparrow})$ is mapped to $(x_{m\uparrow}, y_{m\downarrow})$.
$ior_m = +1$	Do not rotate. A point at $(x_{b\downarrow}, y_{b\downarrow})$ is mapped to $(x_{m\downarrow}, y_{m\downarrow})$ and $(x_{b\uparrow}, y_{b\uparrow})$ is mapped to $(x_{m\uparrow}, y_{m\uparrow})$.

Table 8.12: The effects of the orientation parameters

9 BITMAPS

The bitmap device type is determined by decoding the variable `IBIT` in the `BITMAP_DEVICE` common block. Refer to Table 6.7 on page 12 for the bitmap device codes.

```
INTEGER*4 IBIT
COMMON \BITMAP_DEVICE\ IBIT
```

9.1 The Printronix bitmap

To specify a Printronix type bitmap, set the variable `IBIT` to the value 1. Refer to Table 9.13 on page 28.

IBIT	*dot density (dpi)*	*maximum drawing area (inches)*	*maximum bitmap coordinates*
1	72 in x, 60 in y	12.5×28.44	$0 \leq x_b \leq 2047$ and $0 \leq y_b \leq 749$

Table 9.13: The Printronix bitmap

The Printronix is a bitmap graphics device with a page size of 11 inches in the x-direction by 15 inches in the y-directon. The dot density is 72 dots/inch in the x-direction and 60 dots/inch in the y-direction. The Printronix 'dot' is a rectangle 1.2 times longer in the y-direction than in the x-direction. The bitmap coordinates are restricted to $0 \leq x_b \leq 2047$ and $0 \leq y_b \leq 749$. The maximum drawing area is 12.5×28.44 inches, with $1\frac{1}{4}$ inch margins along the page perforations. Refer to Figure 9.5.

Examples

The following program segment sets up a landscape image which is 12.5×10 inches, with world units of $0 \leq x_w \leq 12.5$ and $0 \leq y_w \leq 10$,

```
IBIT = 1
CALL HARDCOPY_RANGE(0.,719.,0.,749.,0.,12.5,0.,10.,-1)
```

while the following sets up a portrait image which is 10×12.5 inches, with world units of $0 \leq x_w 10$ and $0 \leq y_w 12.5$,

```
IBIT = 1
CALL HARDCOPY_RANGE(0.,719.,0.,749.,0.,10.,0.,12.5,+1)
```

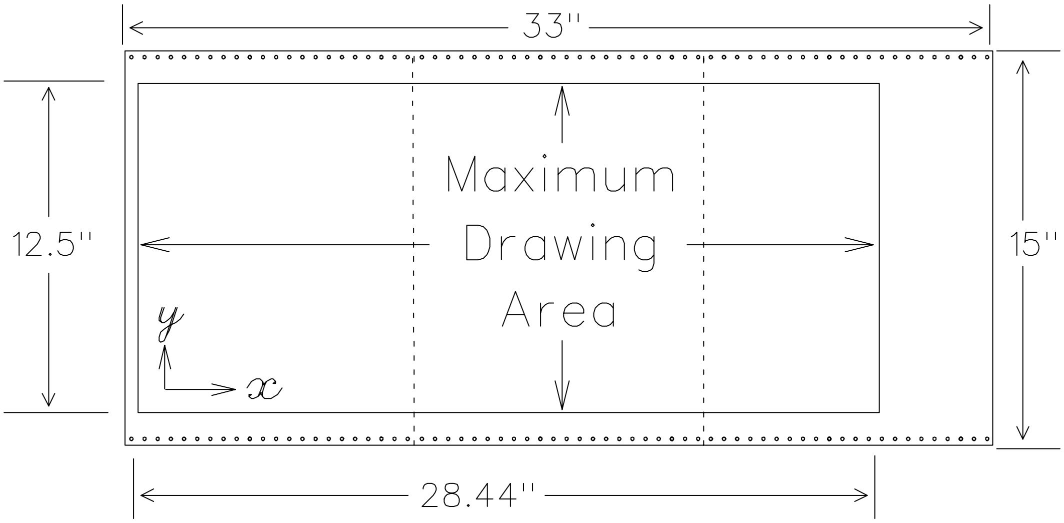


Figure 9.5: Printronix maximum drawing area

9.2 The HP LaserJet bitmap

To specify an HP LaserJet type bitmap, set the variable `IBIT` in the `BITMAP_DEVICE` common block to the value corresponding to the dot density that you want. Refer to Table 9.14 on page 29.

IBIT	*dot density (dpi)*	*maximum drawing area (inches)*	*maximum bitmap coordinates*
2	100	7.5×10	$0 \leq x_b \leq 999$ and $0 \leq y_b \leq 749$
12	150	7.5×10	$0 \leq x_b \leq 1499$ and $0 \leq y_b \leq 1124$
32	300	7.5×10	$0 \leq x_b \leq 2999$ and $0 \leq y_b \leq 2249$

Table 9.14: The HP LaserJet bitmap

The HP LaserJet is a bitmap graphics device with a page size of 11 inches in the x-direction by 8.5 inches in the y-directon. The dot density is the same in both directions. The available drawing area on a page is 7.5×10 inches, see Figure 9.6. There are margins on all four sides of the page.

Examples

The following program segment sets up a 100 dots/inch landscape image on the HP LaserJet which is 10×7.5 inches, with world units of $0 \leq x_w \leq 10$ and $0 \leq y_w \leq 7.5$

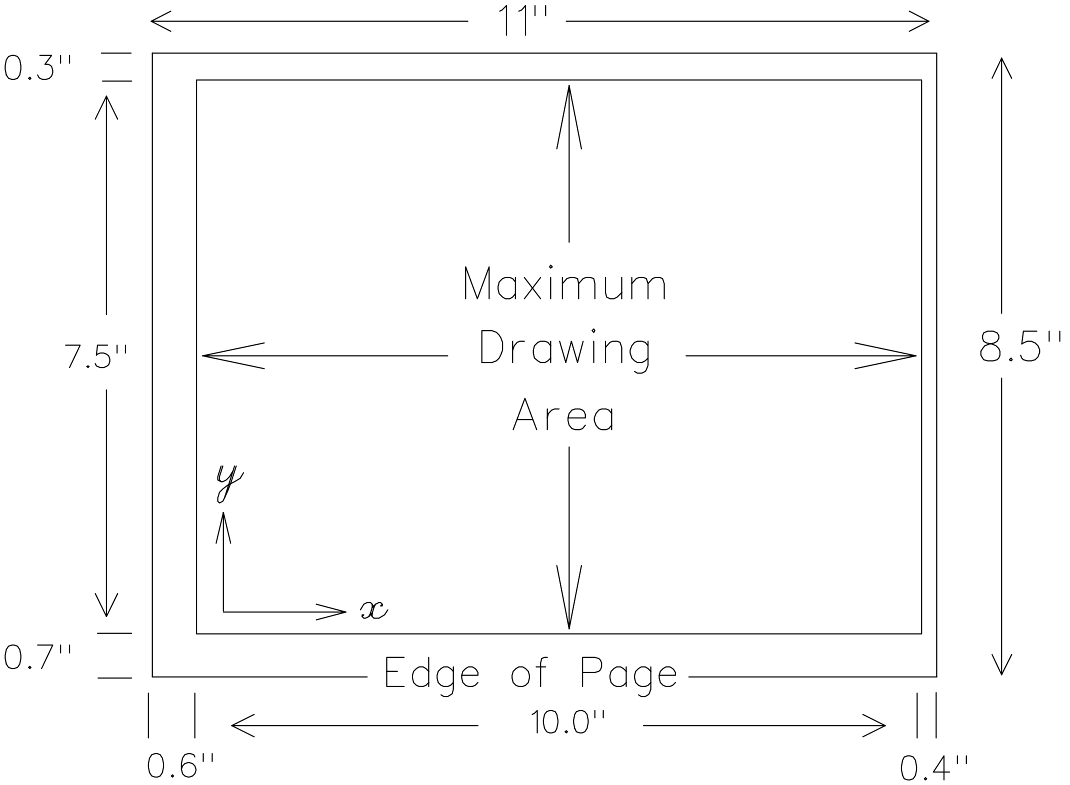


Figure 9.6: HP LaserJet maximum drawing area

```
IBIT=2
CALL HARDCOPY_RANGE(0.,999.,0.,749.,0.,10.,0.,7.5,+1)
```

while the following sets up a 100 dots/inch portrait image which is 7.5×10 inches, with world units of $0 \leq x_w \leq 7.5$ and $0 \leq y_w \leq 10$

```
IBIT=2
CALL HARDCOPY_RANGE(0.,999.,0.,749.,0.,7.5,0.,10.,-1)
```

The following program segment sets up a 150 dots/inch landscape image on the HP LaserJet which is 10×7.5 inches, with world units of $0 \leq x_w \leq 10$ and $0 \leq y_w \leq 7.5$

```
IBIT=12
CALL HARDCOPY_RANGE(0.,1499.,0.,1124.,0.,10.,0.,7.5,+1)
```

The following program segment sets up a 150 dots/inch portrait image which is 7.5×10 inches, with world units of $0 \leq x_w \leq 7.5$ and $0 \leq y_w \leq 10$

```
IBIT=12
CALL HARDCOPY_RANGE(0.,1499.,0.,1124.,0.,7.5,0.,10.,-1)
```

The following program segment sets up a 300 dots/inch landscape image on the HP LaserJet which is 10×7.5 inches, with world units of $0 \leq x_w \leq 10$ and $0 \leq y_w \leq 7.5$

```
IBIT=32
CALL HARDCOPY_RANGE(0.,2999.,0.,2249.,0.,10.,0.,7.5,+1)
```

The following program segment sets up a 300 dots/inch portrait image which is 7.5×10 inches, with world units of $0 \leq x_w \leq 7.5$ and $0 \leq y_w \leq 10$

```
IBIT=32
CALL HARDCOPY_RANGE(0.,2999.,0.,2249.,0.,7.5,0.,10.,-1)
```

9.3 The InkJet colour bitmap

Both the HP PaintJet and the DEC LJ250 are colour bitmap graphics devices with a plot density of 180 dots per inch. The page size is 8.5×11 inches, but the available drawing area is 8×11 inches, that is, there are $\frac{1}{4}$ inch margins on the paper sides which are perpendicular to the page perforations. Multiple page drawings are possible, with a maximum of 5 pages. This gives a maximum drawing area of 8×55 inches, see Figure 9.7 on page 32.

To make use of this device, you must use the `BITMAP_DEVICE` common block.

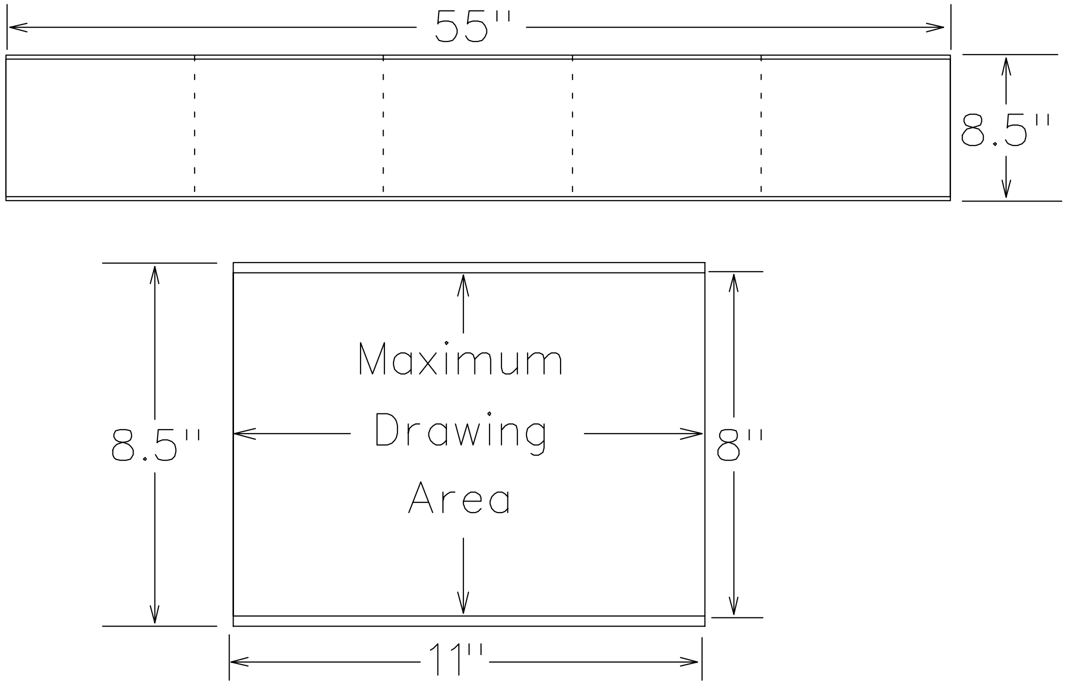


Figure 9.7: Maximum InkJet drawing area

```
INTEGER*4 IBIT
COMMON \BITMAP_DEVICE\ IBIT
```

and set `IBIT = 5` before calling `HARDCOPY_RANGE`.

When the InkJet type of bitmap is chosen, the memory space for the bitmap is dynamically allocated and is not subject to the restraints as mentioned in the **Bitmap Coordinates** section.

The InkJet bitmap coordinates, are restricted to $0 \leq x_b \leq 9899$ and $0 \leq y_b \leq 1439$. Multiple pages are selected by choosing the correct value for $x_b = 180 \times 11 \times np$, where np is the number of pages required, $1 \leq np \leq 5$.

9.4 Turning the bitmap off/on

Output to the bitmap can be turned off by using the `TO_BIT_OR_NOT` common block.

```
LOGICAL*4 WELL
COMMON /TO_BIT_OR_NOT/ WELL
```

If `WELL` is true, then graphics output is directed to the bitmap.
If `WELL` is false, graphics output is not put in the bitmap.

9.5 Turning the monitors off/on

The monitor type and the output unit number for the first monitor can be found in the `PLOTMONITOR` common block.

```
INTEGER*4 IMONITOR, IOUTM
COMMON /PLOTMONITOR/ IMONITOR, IOUTM
```

where `IMONITOR` $= t_1$ and `IOUTM` $= u_1$, as defined in the call to `MONITOR_RANGE`.

The monitor type and the output unit number for the second monitor can be found in the `PLOTMONITOR2` common block.

```
INTEGER*4 IMONITOR2, IOUTM2
COMMON /PLOTMONITOR2/ IMONITOR2, IOUTM2
```

where `IMONITOR2` $= t_2$ and `IOUTM2` $= u_2$, as defined in the call to `MONITOR2_RANGE`.

These common blocks can be used for temporarily turning off a monitor, that is, stopping output to a monitor, by setting `IMONITOR` or `IMONITOR2` to zero.

9.6 HP LaserJet bitmap and PostScript example

The following example program uses calls to the fundamental control routines that are equivalent to the defaults, except that a second monitor is added, a PostScript plot file type monitor using A size paper, see Table 7.10 on page 20. Assuming that the printer resolution is 300 dots/inch, the second monitor viewport is defined to be $150 \Rightarrow 3102$ in x, which gives 9.84 inches with $\frac{1}{2}$ inch margins on both sides; and $150 \Rightarrow 2394$ in y, which gives 7.48 inches with $\frac{1}{2}$ inch margins on both sides. The first monitor is an X window device.

Note: Call `HARDCOPY_RANGE` first, and call `CLEAR_PLOT` to initialize the drawing.

```
CALL HARDCOPY_RANGE(0.,479.,0.,639.,0.,639.,0.,479.,-1)
CALL MONITOR_RANGE(18,6,0.,479.,0.,639.,0.,1.,0.,.75,-1)
CALL MONITOR2_RANGE(14,7,0.,479.,0.,639.,150.,3102.,150.,2394.,-1)
CALL CLEAR_PLOT
CALL PLOT_R(638.,15.,3)
CALL PLOT_R(638.,1.,2)
CALL PLOT_R(1.,1.,2)
CALL PLOT_R(1.,478.,2)
CALL PLOT_R(15.,478.,2)
CALL PSYM(250.,3.,20.,'BOTTOM',0.,6)
CALL PSYM(25.,200.,20.,'LEFT',90.,4)
CALL FLUSH_PLOT
CALL NARGSI(1)
CALL GRAPHICS_HARDCOPY(0)
END
```

The above program produces hardcopies as shown in Figure 9.8 on page 34. The picture on the terminal is similar to the PostScript output. The bitmap output is a landscape picture on a portrait page, while the PostScript output is a landscape picture on landscape page.

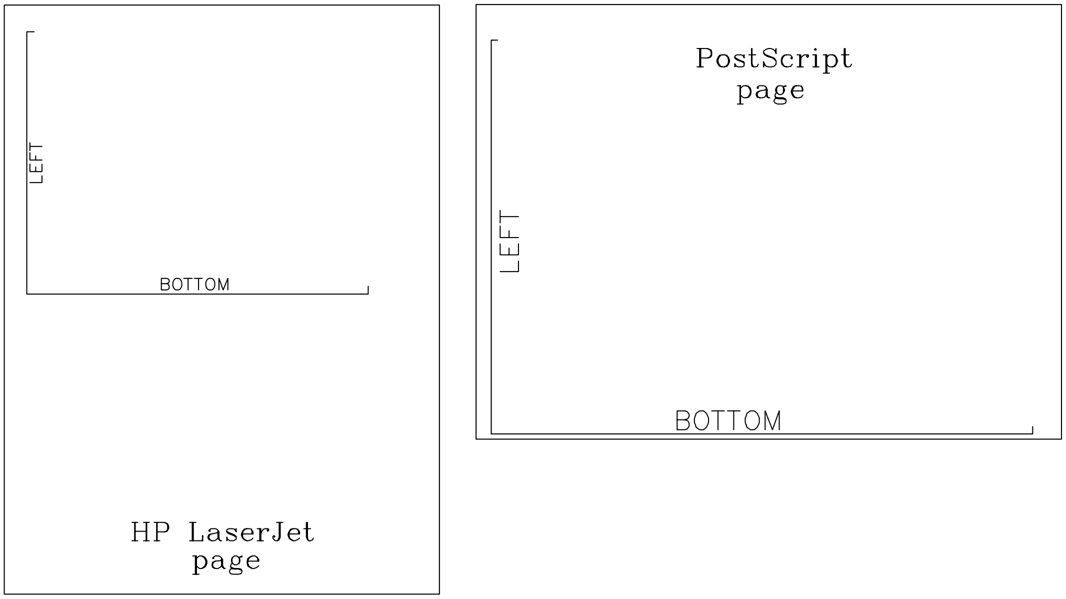


Figure 9.8: An example illustrating the default windows and viewports

9.7 Commensurate drawing example

Suppose you want to draw commensurate objects, for example, circles should appear as circles, on the monitor screen, the HP LaserJet and PostScript. The following sample program produces commensurate drawings on an X window device, an HP LaserJet, and an PostScript.

```
XMAX  = 19.0 / 2.54 ! convert cm to inches
YMAX  = 25.0 / 2.54 ! convert cm to inches
```

```
XMAXP = 150.*XMAX   ! 150 dots/inch
YMAXP = 150.*YMAX   ! 150 dots/inch
CALL HARDCOPY_RANGE(0.,XMAXP,0.,YMAXP,0.,XMAX,0.,YMAX,-1)
XMN  = (1.-.75*XMAX/YMAX)/2.
XMX  = (1.+.75*XMAX/YMAX)/2.
CALL MONITOR_RANGE(18,6,0.,XMAXP,0.,YMAXP,XMN,XMX,0.,0.75,-1)
XMAXM2 = FLOAT(INT((YMAX+0.5)*300.)) ! 300dpi with 1/2in margins
YMAXM2 = FLOAT(INT((XMAX+0.5)*300.)) ! 300dpi with 1/2in margins
CALL MONITOR2_RANGE(14,7,0.,XMAXP,0.,YMAXP,150.,XMAXM2,150.,YMAXM2,1)
RADIUS = 2.5
X = RADIUS+(XMAX/2.)
Y = (YMAX/2.)
CALL PLOT_R(X,Y,3)
DO I = 1, 98
  X = RADIUS*COSD(I*180./49.)+(XMAX/2.)
  Y = RADIUS*SIND(I*180./49.)+(YMAX/2.)
  CALL PLOT_R(X,Y,2)
END DO
HEIGHT = 0.5
X = (XMAX/2.)-3.*HEIGHT*0.75
Y = 0.1
CALL PSYM(X,Y,HEIGHT,'BOTTOM',0.,6)
X = HEIGHT*1.5
Y = (YMAX/2.)-2.*HEIGHT*0.75
CALL PSYM(X,Y,HEIGHT,'LEFT',90.,4)
CALL FLUSH_PLOT
CALL NARGSI(1)
CALL GRAPHICS_HARDCOPY(0)
STOP
END
```

The above program produces hardcopies shown in Figure 9.9 on page 36. The graphics is in portrait mode.

9.8 Batch example

Suppose you are writing a program to run as a batch job, so you do not want terminal output. The following sample program produces the same drawing as in the previous example, but there is no terminal output.

```
CHARACTER*20 QUE_NAME
COMMON /QUE_NAMES/ QUE_NAME
QUE_NAME = 'PLT_119'
```

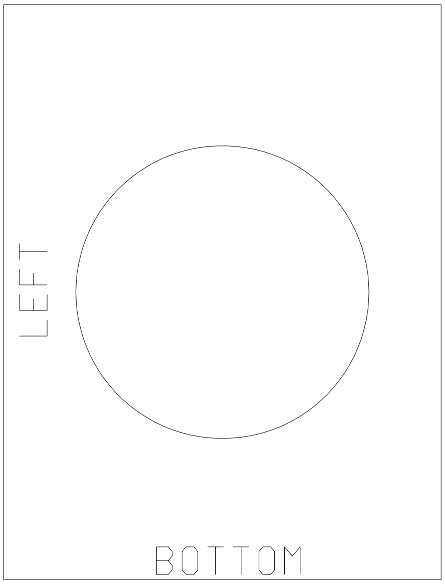


Figure 9.9: An example illustrating commensurate drawings

```
CALL HARDCOPY_RANGE(0.,1499.,0.,1124.,0.,10.0,0.,7.5,+1)
CALL MONITOR_RANGE(5,7,0.,1499.,0.,1124.,0.,25.40,0.,19.05,+1)
CALL CLEAR_PLOT
RADIUS = 3.0
CALL PLOT_R(8.0,3.75,3)
DO I = 1, 98
  X = RADIUS*COSD(I*180./49.)+5.0
  Y = RADIUS*SIND(I*180./49.)+3.75
  CALL PLOT_R(X,Y,2)
END DO
CALL PSYM(3.5,0.1,0.5,'BOTTOM',0.,6)
CALL PSYM(0.75,3.0,0.5,'LEFT',90.,4)
CALL FLUSH_PLOT
CALL NARGSI(1)
CALL GRAPHICS_HARDCOPY(-3)
END
```

`MONITOR_RANGE` is called with the same parameters as `MONITOR2_RANGE` in the previous example and `MONITOR2_RANGE` is not called at all. This means that the first monitor will be the HP pen plotter, and there is no second monitor.

`GRAPHICS_HARDCOPY` is called with -3 to indicate that the hardcopy is to be automatically submitted to an HP pen plotter queue. The `QUE_NAMES` common block setting means that the drawing will be automatically submitted to the `PLT_119` queue. The `NARGSI` routine is required to indicate the number of parameters with which the `GRAPHICS_HARDCOPY` routine has been called.

9.9 InkJet example

Suppose you want to obtain 2 pages of output on an InkJet colour bitmap device, and you want to preview drawings on an X window device. You want the drawing centred with commensursate scaling. To make an InkJet drawing, you must use the `BITMAP_DEVICE` common block variable `IBIT` before calling `HARDCOPY_RANGE`. The following sample program produces commensurate drawings on an X window device and an InkJet plotter.

```
INTEGER*4 IBIT
COMMON /BITMAP_DEVICE/ IBIT
IBIT  = 5
PAGES = 2.
YMAXP = 180. *  8.            ! 180dpi * 8.0in (0.25in margins)
XMAXP = 180. * 11. * PAGES    ! 180dpi * 11in * number_of_pages
XMN   = 0.
XMX   = 1.
YMN   = (0.75 - 1. * 8. / 11. / PAGES) / 2.
YMX   = (0.75 + 1. * 8. / 11. / PAGES) / 2.
CALL HARDCOPY_RANGE(0.,XMAXP,0.,YMAXP,0.,PAGES*11.,0.,8.,+1)
CALL MONITOR_RANGE(18,6,0.,XMAXP,0.,YMAXP,XMN,XMX,YMN,YMX,+1)
CALL CLEAR_PLOT
RADIUS = 3.0
CALL PLOT_R(8.5,4.0,3)
DO I = 1, 98
  X = RADIUS*COSD(I*180./49.)+5.5
  Y = RADIUS*SIND(I*180./49.)+4.0
  CALL PLOT_R(X,Y,2)
END DO
CALL PSYM(4.5,0.1,0.5,'PAGE 1',0.,6)
CALL PSYM(0.75,3.0,0.5,'LEFT',90.,4)
CALL PLOT_R(19.5,4.0,3)
DO I = 1, 98
  X = RADIUS*COSD(I*180./49.)+16.5
  Y = RADIUS*SIND(I*180./49.)+4.0
  CALL PLOT_R(X,Y,2)
END DO
CALL PSYM(15.5,0.1,0.5,'PAGE 2',0.,6)
CALL PSYM(11.75,3.0,0.5,'LEFT',90.,4)
CALL FLUSH_PLOT
CALL NARGSI(1)
CALL GRAPHICS_HARDCOPY(0)
END
```

The above program produces an InkJet hardcopy as shown in Figure 9.10 on page 38.

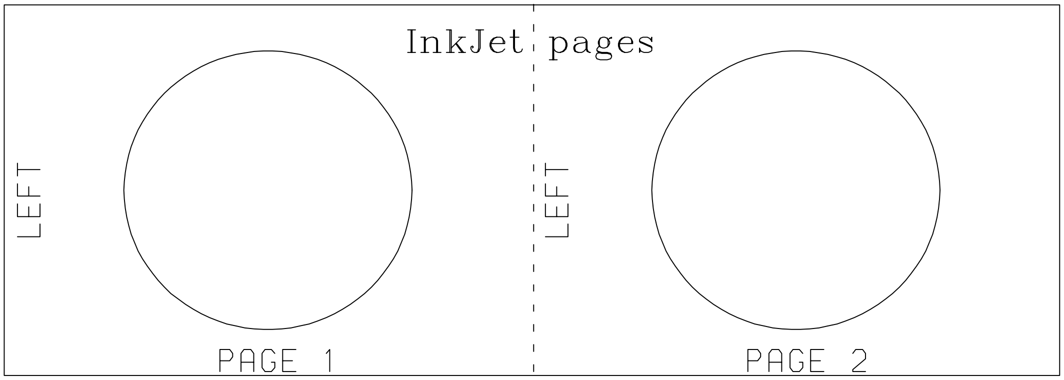


Figure 9.10: An example illustrating InkJet drawings

9.10 SET_PLOT_DEVICES

Instead of explicitly specify the viewing transformations, one can simply select a pre-defined device configuration by calling:

subroutine `SET_PLOT_DEVICES(` t_1, u_1, t_2, u_2, $ibit$, $units$, $mode$, np, `*,*,*,*,*` `)`

	variable	*type*	*description*
input	t_1	`INTEGER*4`	monitor 1 type code
	u_1	`INTEGER*4`	monitor 1 logical unit number
	t_2	`INTEGER*4`	monitor 2 type code
	u_2	`INTEGER*4`	monitor 2 logical unit number
	$ibit$	`INTEGER*4`	bitmap device code
	$units$	`CHARACTER*2`	world units type
	$mode$	`CHARACTER*(*)`	orientation
	np	`INTEGER*4`	number of InkJet pages
output			

The input parameters' permissable values are shown in Table 9.15 on page 39, while the meanings of the alternate returns are given in Table 9.16 on page 40.

To properly initialize the graphics routines, you should call `CLEAR_PLOT` after a call to the `SET_PLOT_DEVICES` routine.

parameter	*data type*	*values*	*results*
t_1	`INTEGER*4`	0	non-graphics terminal monitor
		1	VT640
		2	Tek 40xx
		6	Citoh CIT467
		7	Tek 41xx/42xx
		8	Regis VT241
		9	Plessey PT100G
		12	Seiko GR1105
		17	generic
		18	X window
u_1	`INTEGER*4`	$1 - 99$	first monitor output unit (usually 6)
t_2	`INTEGER*4`	0	no second monitor
		1	display file – VT640 type
		2	display file – Tek 40xx type
		5	HP pen plotter
		6	display file – Citoh CIT467 type
		7	display file – Tek 41xx/42xx type
		8	display file – Regis VT241 type
		9	display file – Plessey PT100G type
		11	Houston Inst. pen plotter
		12	display file – Seiko GR1105 type
		13	Imagen
		14	PostScript
		16	LN03+
		19	GKS metafile
		20	Roland GL pen plotter
u_2	`INTEGER*4`	$1 - 99$	second monitor output unit (usually 7)
ibit	`INTEGER*4`	0	no bitmap
		01	Printronix bitmap
		02	HP LaserJet (100 dpi) bitmap
		12	HP LaserJet (150 dpi) bitmap
		32	HP LaserJet (300 dpi) bitmap
		03	HP ThinkJet bitmap
		04	LA100 bitmap
		05	InkJet colour bitmap
units	`CHARACTER*2`	`'CM'`	world units to be in centimeters
		`'IN'`	world units to be in inches
mode	`CHARACTER*(*)`	`'LANDSCAPE'` `'PORTRAIT'`	orientation
np	`INTEGER*4`	$1 - 5$	number of pages of InkJet output if $ibit = 5$ then $1 \leq np \leq 5$ if $ibit \neq 5$ then np is assumed to be $= 1$

Table 9.15: `SET_PLOT_DEVICES` calling parameters

RETURN1	invalid $mode$
RETURN2	invalid t_1
RETURN3	invalid t_2
RETURN4	invalid $ibit$
RETURN5	invalid $units$

Table 9.16: `SET_PLOT_DEVICES` alternate returns

Pre-defined world coordinate systems

The `SET_PLOT_DEVICES` routine guarantees that the first monitor, the second monitor, and the bitmap hardcopy viewports all will have a commensurate aspect ratio, that is, circles will appear as circles on all devices.

The world coordinate system horizontal and vertical ranges are determined by the second monitor and are shown in Table 9.17 on page 41. The exception to this rule is when the InkJet is chosen as the bitmap device by calling `SET_PLOT_DEVICES` with $ibit = 5$. In this case, the world coordinate system units range is 8×11 inches (20.32×27.94 centimetres).

Choosing paper sizes

The default paper size is A size. To choose a different paper size, use the `PLOTTER_SIZE` common block.

```
INTEGER*4 PLTR_SIZE
COMMON /PLOTTER_SIZE/ PLTR_SIZE
```

Set the `PLTR_SIZE` variable to the appropriate value, as shown in Table 9.18 on page 42, *before* calling `SET_PLOT_DEVICES`. By default, the variable `PLTR_SIZE = 0`.

Paper size applies to HP, Houston Instruments and Roland GL pen plotters, as well as PostScript, but not all devices accept all these sizes of paper. The user must check this beforehand and sometimes manually load the odd sizes of paper.

Display files

Display files are graphics files that can be simply typed on the appropriate terminal to redisplay the graphics.

If a display file is chosen as the second monitor, for example, $t_2 = 1$, then the world coordinate

		range of world coordinates					
		centimeters			*inches*		
HP	size A	25.00	×	19.00	9.84	×	7.48
	size B	40.64	×	25.40	16.00	×	10.00
	size C	53.34	×	40.64	21.00	×	16.00
	size D	83.82	×	53.34	33.00	×	21.00
	size E	109.22	×	83.22	43.00	×	33.00
Houston Inst.	size A	25.00	×	19.00	9.84	×	7.48
	size B	40.64	×	25.40	16.00	×	10.00
	size C	53.34	×	40.64	21.00	×	16.00
	size D	83.82	×	53.34	33.00	×	21.00
	size E	109.22	×	83.22	43.00	×	33.00
Imagen		27.94	×	21.59	11.00	×	8.50
PostScript	size A	25.00	×	19.00	9.84	×	7.48
	size B	40.64	×	25.40	16.00	×	10.00
	size C	53.34	×	40.64	21.00	×	16.00
	size D	83.82	×	53.34	33.00	×	21.00
	size E	109.22	×	83.22	43.00	×	33.00
	size A3	25.40	×	19.05	10.00	×	7.50
	size A4	27.16	×	18.46	10.69	×	7.27
LN03+		27.94	×	21.59	11.00	×	8.50
GKS metafile		27.94	×	21.59	11.00	×	8.50
Roland GL	size A	24.94	×	16.19	9.82	×	6.37
	size B	37.78	×	24.94	14.87	×	9.82
	size C	52.88	×	37.78	20.82	×	14.87
	size D	83.36	×	50.48	32.82	×	19.87
	size E	106.36	×	83.36	41.87	×	32.82

Table 9.17: World coordinate ranges obtained with `SET_PLOT_DEVICES`

`PLTR_SIZE`	*paper size*
0	A
1	B
2	C
3	D
4	E
5	A3
6	A4

Table 9.18: Choosing paper size with `SET_PLOT_DEVICES`

system ranges will be 8.5×11 inches (21.59×27.94 centimetres).

10 FUNDAMENTAL DRAWING ROUTINES

Having set up the viewing transformations, monitor types and logical input/output units, line segments and points can be drawn with PLOT_R, while text can be drawn with PSYM. Together with the device drivers that they invoke, these are the most basic drawing routines in the graphics package. They control a virtual pen that simultaneously moves over the bitmap and monitors, and sends instructions to the drawing editor, if an **EDGR** file is active.

10.1 Line segments and points

Drawing a line segment requires a means of moving a virtual pen to a specified location with the pen up, to start a line segment, and down, to draw the line segment. Drawing a point means to move with the pen up to a specified location, put the pen down, and then to put the pen back up.

subroutine PLOT_R(x, y, $ipen$)

	variable	*type*	*description*
input	x	REAL*4	world coordinate
	y	REAL*4	world coordinate
	$ipen$	INTEGER*4	virtual pen code
output			

x and y are the world coordinates to which the virtual pen will move, and $ipen$ is the code that controls the virtual pen movement. Refer to Table 10.19 on page 43. Prior to the first invocation of PLOT_R, the default virtual pen position is $(0, 0)$.

$ipen$	*result*
2	draw from the current position to location (x, y)
3	move to location (x, y) without drawing
20	draw a point at location (x, y)

Table 10.19: Pen code for routine PLOT_R

The last world coordinates at which the virtual pen was placed, and the pen code that was last used, can be found in the PLOT_PREVIOUS common block.

```
REAL*4    XP, YP
INTEGER*4 IPENP
COMMON /PLOT_PREVIOUS/ XP, YP, IPENP
```

where XP $= x$, YP $= y$ and IPENP $= ipen$, as defined in the last call to PLOT_R.

Fundamental Drawing Routines

Dashed lines

Dashed lines can be drawn with the `DLINE` routine, which calls the `PLOT_R` routine for you.

subroutine `DLINE(` x, y, $ltyp$, $iend$ `)`

	variable	*type*	*description*
input	x	`REAL*4`	world coordinate
	y	`REAL*4`	world coordinate
	$ltyp$	`INTEGER*4`	line type
	$iend$	`INTEGER*4`	termination code
output			

This routine draws a line of specified line type, $ltyp$, from the current location to location (x, y) in the world coordinate system. There are ten (10) line types available at any time, and, by default, come in four different styles:

- ordinary solid line
- double line
- dashed line with one dash length and one space length
- dashed line with two dash lengths and one space length

See Figure 10.11 on page 44 for examples of the default line types.

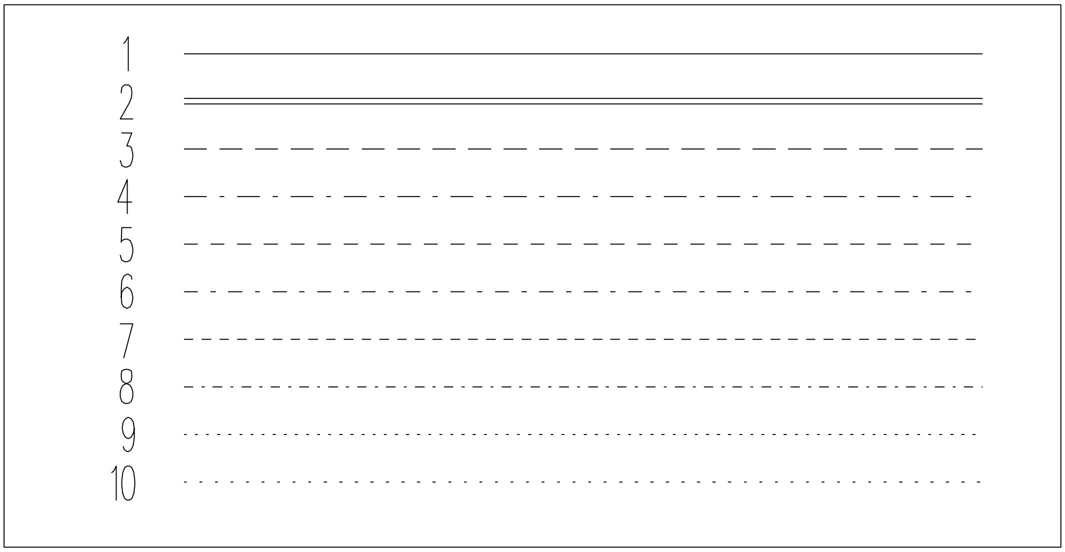


Figure 10.11: Examples of the ten default line types

The $iend$ parameter controls how the dashing sequence is terminated at the (x, y) location.

$iend$	*result*
0	draw an intermediate line segment, that is, one that is to be connected to a succeeding line segment.
1	end the dashing sequence at (x, y). If the sequence ends on a space, the previous dash will be extended to (x, y), provided this does not lengthen the dash by more than 100%. This improves the appearance for some types of plotting. A new dashing sequence will begin on the next call to `DLINE`.
2	end the dashing sequence at (x, y), but unconditionally extend the last dash to (x, y).

The routine `DLINE` always plots from the current location, which is generally determined by the last call to `PLOT_R`. Thus a call to `PLOT_R` with pen up would usually precede a series of calls to `DLINE`. In order to plot a continuous dashed line through a series of points, `DLINE` "remembers" where it is at between calls. As long as the current location is the same as that where `DLINE` left off on the previous call, and the line type, $ltyp$, has not changed, `DLINE` will continue dashing from where it left off. Thus, the dashing may be interrupted, a symbol or other items plotted, and the dashing can be continued by calling `PLOT_R` with the pen up, that is, with $ipen$ set to 3, to the old location before calling `DLINE` again.

10.1.0.1 Defining line types

For each line type, $ltyp$, the appearance of the line is determined by three parameters p_1, p_2, and p_3 that are stored in an internal table. See Table 10.20 on page 45 for details. The values for p_1, p_2, and p_3 should be in the world coordinate system of units.

p_1	p_2	p_3	*Result*
$= 0$			ordinary solid line
> 0	$= 0$		double line with width p_1
> 0	> 0	$= 0$	dashed line with dash length p_1, space length p_2
> 0	> 0	> 0	dashed line with first dash length p_1, space length p_2, and second dash length p_3

Table 10.20: Line type definitions used by `DLINE`

The user may define line types, or use the default types established in this routine, which are suitable for the default 640×480 world coordinate system. The default line types, as shown in Figure 10.11 on page 44, are detailed in Table 10.21 on page 46.

$ltyp$	p_1	p_2	p_3
1	0	0	0
2	2	0	0
3	30	10	0
4	30	15	10
5	20	10	0
6	20	8	6
7	10	6	0
8	10	6	4
9	3	10	0
10	5	15	0

Table 10.21: The default line types

Line types can be redefined by the user by calling the `DLINESET` routine.

subroutine	`DLINESET(` $ltyp$, p_1, p_2, p_3 `)`		
	variable	*type*	*description*
input	$ltyp$	`INTEGER*4`	line type
	p_1	`REAL*4`	first line parameter
	p_2	`REAL*4`	second line parameter
	p_3	`REAL*4`	third line parameter
output			

Alternatively, the line types can be scaled to fit a user defined world coordinate system by calling the `DLINESCALE` routine.

subroutine	`DLINESCALE(` $ltyp$, sf `)`		
	variable	*type*	*description*
input	$ltyp$	`INTEGER*4`	line type code
	sf	`REAL*4`	scale factor
output			

For example, if the user defined world coordinate system is 10 units in x by 7.5 units in y, the scale factor: $sf = \frac{10}{640}$ would be appropriate.

10.2 Text

Text includes character strings, integers and real numbers. Text is drawn by using software characters generated by numerous pen strokes. Software characters can have any size or orientation, and are drawn using the `PSYM` routines.

To draw a character string use the `PSYM` routine.

subroutine `PSYM(` x, y, $height$, $string$, $angle$, n `)`

	variable	*type*	*description*
input	x	`REAL*4`	world coordinate
	y	`REAL*4`	world coordinate
	$height$	`REAL*4`	text height
	$string$	`LOGICAL*1`	text array
	$angle$	`REAL*4`	text angle
	n	`INTEGER*4`	length of $string$
output			

(x, y) is the location, in world coordinates, of the lower left hand corner of the first character in $string$. The character height, $height$, should be in world units. $string$ is an ASCII character string passed by reference.[6] The angle of the string, $angle$, should be in degrees, measured counterclockwise from the horizontal. n is the number of characters in the $string$. Text generated by `PSYM` is drawn by pen strokes generated by calls to the `PLOT_R` routine, and thus will be clipped at the world window and then at the bitmap window(s).

As an example, you could use either of the following methods to draw the string 'Look here!' contained in `CHARACTER*20` variable `STR` at world coordinates $(100, 100)$ with a character height of 20 world units and at an angle of $45°$ measured clockwise from the horizontal:

```
CALL PSYM(100.,100.,20.,%REF(STR),-45.,10)
CALL PSYM(100.,100.,20.,'Look here!',-45.,10)
```

To draw an integer or real number, you could use the "internal write" feature of FORTRAN. For example, to draw the real number contained in a `REAL*4` variable `DATUM` in `F5.2` format, first declare a `CHARACTER` variable large enough to hold the formatted `F5.2` output, for example, `CHARACTER*10 NUMB`, and then do the following:

---

[6] There are several ways to pass a character string by reference in VAX FORTRAN. Consider the string 'aabbccdd'. All the calls in the following code fragment would be valid:

```
CHARACTER STR1*8/'aabbccdd'/
LOGICAL*4 STR2(2)/'62626161'X,'64646363'X/
CALL PSYM(0.,0.,10.,'aabbccdd',0.,8)
CALL PSYM(0.,0.,10.,%REF(STR1),0.,8)
CALL PSYM(0.,0.,10.,STR2,0.,8)
```

```
      WRITE(NUMB,30)DATUM
 30   FORMAT(F5.2)
      CALL PSYM(X,Y,HEIGHT,%REF(NUMB),THETA,5)
```

The FORTRAN internal file facility, coupled with PSYM, provides a means of drawing any numeric or character entity in any FORTRAN format.

Fonts

By default, PSYM uses the STANDARD character set. See the TRIUMF GRAPHIC FONTS manual for examples of text font tables. Table 10.22 on page 48 lists the names of the available fonts.

STANDARD	ITALIC.2	GOTHIC.ENGLISH	ROMAN.2
	ITALIC.2A	GOTHIC.FRAKTUR	ROMAN.2A
SANSERIF.1	ITALIC.3	GOTHIC.ITALIAN	ROMAN.3
SANSERIF.2			
SANSERIF.CART	SCRIPT.1	CYRILLIC.2	OLDALPH
	SCRIPT.2		
HELVETICA.1		KATAKANA	KANJI1
	GREEK.1	HIRAGANA	KANJI2
TRIUMF.1	GREEK.2		KANJI3
TRIUMF.2	GREEK.2A		KANJI4
TSAN	GREEK.CART		KANJI5
ROMAN.FUTURA	ROMAN.SWISSL	SPECIAL	HEBREW
ROMAN.SERIF	ROMAN.SWISSM	MATH	
ROMAN.FASHON	ROMAN.SWISSB	TRIUMF.OUTLINE	
ROMAN.LOGO1			

Table 10.22: The font names

To change the character set used by PSYM call the PFONT routine.

subroutine	PFONT(*name*, *what*)		
	variable	*type*	*description*
input	*name*	CHARACTER*(*)	font name
	what	INTEGER*4	flag
output			

what is a flag which should be 0 if *name* is the name of a library character set, and non-zero if it is a file name for a user-defined set.

VMS

If a library character set is specified, it is read from `TRIUMF$FONTS:VAXFONT.DAT`, where `TRIUMF$FONTS` is a logical name. For example:

```
$ DEFINE TRIUMF$FONTS DISK:[TRIUMF.GPLOT]
```

UNIX

If a library character set is specified, it is read from `TRIUMF_FONTS:vaxfont.dat`, where `TRIUMF_FONTS` is an environment variable. For example:

```
% setenv TRIUMF_FONTS /usr/users/triumf/gplot
```

Special characters

A *special character* is any character that is not available on your terminal's keyboard. Use the `MODE` common block to draw such a special character.

```
LOGICAL*4 ASIS
COMMON /MODE/ ASIS
```

The default value for `ASIS` is `FALSE`. There are six special characters which produce carriage returns, backspaces, subscripts, and superscripts. Refer to Table 10.23 on page 50. The first character shown for each is used if `ASIS = .TRUE.` in the `MODE` common block, that is, if the input *string* is in EBCDIC code, the native code of the `PSYM` font structure. The second character shown is used if `ASIS = .FALSE.`, the default, that is, if the input *string* is in ASCII code. In this case the *string* will be converted to EBCDIC code by a translation routine.

The exact location and height of subscripts and superscripts are determined by the particular alphabet. Multiple levels of subscripts and superscripts are permitted; all levels are drawn with the same height for `'09'X` and `'38'X`, and they are drawn with different height for `'0A'X` and `'39'X`. When change of size occurs, it is always a fraction of *height*.

Other non-keyboard characters can also be drawn. For example, to draw the the Ξ symbol from the `TRIUMF.2` font:

1. set `ASIS` to `.TRUE.`;
2. call `PSYM` with the hexadecimal code representing the special character, for example, the Ξ symbol in the `TRIUMF.2` font can be drawn with: `CALL PSYM(X,Y,HEIGHT,'B0'X,ANGLE,1)`
3. reset `ASIS` to `.FALSE.`

Length of drawn text

EBCDIC	ASCII	*result*
`'15'X`	`'0A'X`	Carriage return. This causes an immediate carriage return as on a typewriter, where the margin is defined by the last previous carriage return, or the last explicit (x, y) pair.
`'16'X`	`'08'X`	Backspace. This produces a one character backspace. If the alphabet currently being used has variable character spacing, this may be farther than one character back.
`'09'X`	`'09'X`	Up. If encountered while drawing text normally, succeeding text is drawn as superscripts. If encountered while drawing superscripts, text is drawn at the next higher superscript level. If encountered while drawing subscripts, drawing continues at the next higher level of subscripts, or back to normal if there was only one level of subscripts.
`'38'X`	`'01'X`	Down. If encountered while drawing text normally, succeeding text is drawn as subscripts. If encountered while drawing subscripts, text is drawn at the next lower subscript level. If encountered while drawing superscripts, succeeding text is drawn at the next lower level of superscripts, or back to normal if there was only one level of superscripts.
`'0A'X`	`'03'X`	Up. Similar to `'09'X`, but the height is always changed.
`'39'X`	`'02'X`	Down. Similar to `'38'X`, but the height is always changed.

Table 10.23: `PSYM` special characters

To determine the length in world units that a text string would have if drawn by `PSYM`, use the `PSMLEN` function.

`REAL*4` *function* `PSMLEN(` $string$, n, $height$ `)`

	variable	*type*	*description*
arguments	$string$	`LOGICAL*1`	text array
	n	`INTEGER*4`	length of $string$
	$height$	`REAL*4`	text height

The `PSMLEN` function returns the length of $string$, an ASCII character string, passed by reference, having n characters, with text $height$ expressed in world coordinate units.

10.3 Hatch patterns

Hatch patterns may be used to fill polygonal regions. Hatch fill patterns are drawn with the `HATCH_DRAW` routine.

subroutine `HATCH_DRAW(` $xpol$, $ypol$, $nvert$, $nhatch$ `)`

	variable	*type*	*description*
input	$xpol$	`REAL*4`	vertex coordinate array
	$ypol$	`REAL*4`	vertex coordinate array
	$nvert$	`INTEGER*4`	number of vertices, $nvert < 1000$
	$nhatch$	`INTEGER*4`	pattern number, $1 \leq nhatch \leq 10$
output			

$xpol$ and $ypol$ are arrays containing the (x, y) world coordinates of the vertices of the polygon. The maximum number of vertices allowed is 1000. There are ten (10) hatch pattern types available at any time, $1 \leq nhatch \leq 10$.

A hatch pattern is composed of an angle and one to ten spacings. These spacings are cycled through as each polygon is being filled, that is, a line is drawn inside the polygon at the specified angle, then a parallel line is drawn at the first spacing, then another parallel line is drawn at the second spacing, and so on for the number of spacings in that pattern. This process is repeated until the polygon is filled.

Defining hatch patterns

There are ten hatch patterns available. The default patterns are listed in Table 10.24 on page 53. See Figure 10.12 on page 52 for examples. The spacings are expressed in the default world coordinate system units, $0 \leq x \leq 639$ and $0 \leq y \leq 479$, and the angles are in degrees.

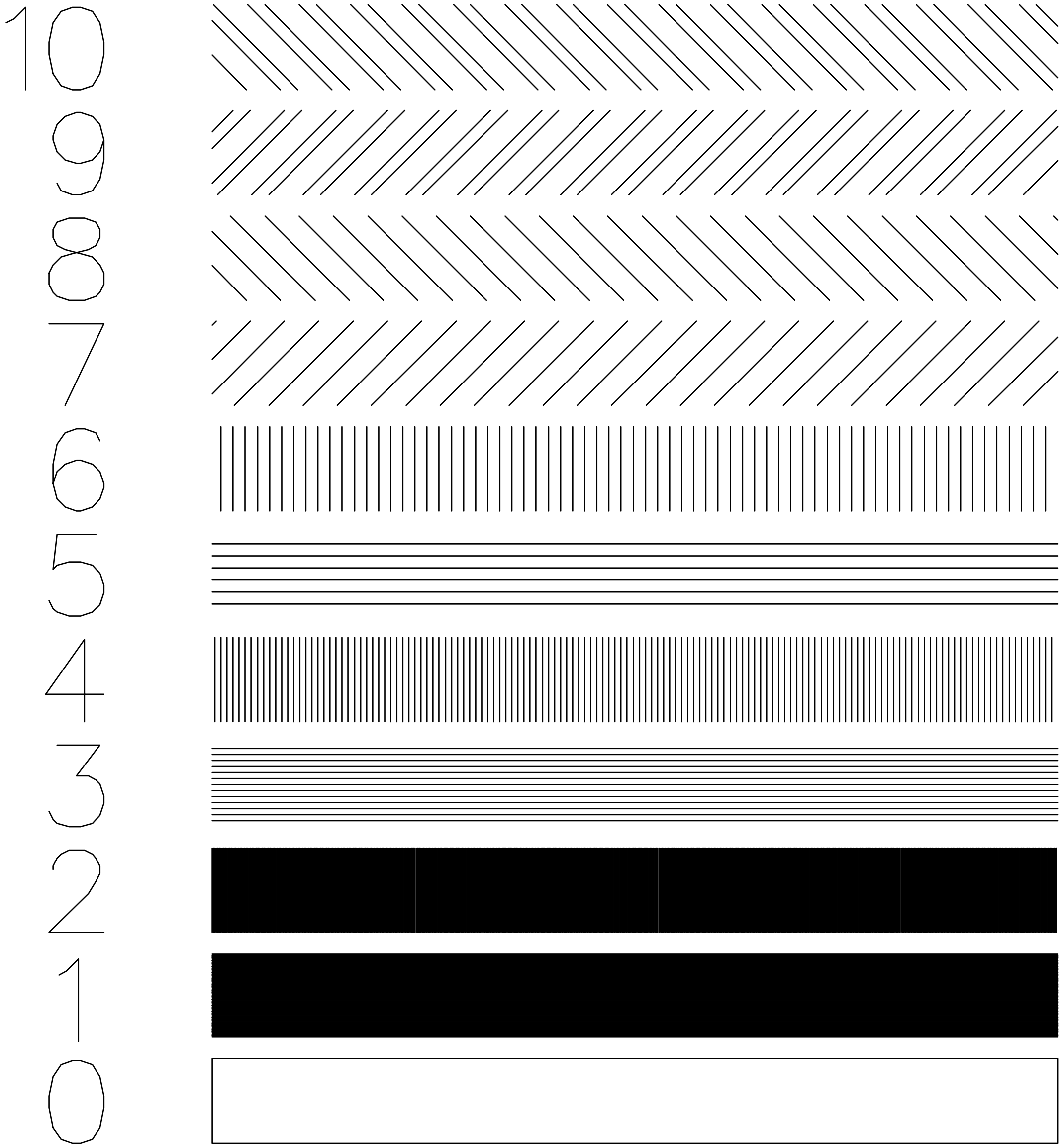


Figure 10.12: Examples of the fill patterns obtained with `HATCH_DRAW`

pattern	*spacing numbers*										
number	1	2	3	4	5	6	7	8	9	10	*angle*
1	1	—	—	—	—	—	—	—	—	—	0
2	1	—	—	—	—	—	—	—	—	—	90
3	5	—	—	—	—	—	—	—	—	—	0
4	5	—	—	—	—	—	—	—	—	—	90
5	10	—	—	—	—	—	—	—	—	—	0
6	10	—	—	—	—	—	—	—	—	—	90
7	20	—	—	—	—	—	—	—	—	—	45
8	20	—	—	—	—	—	—	—	—	—	−45
9	20	10	—	—	—	—	—	—	—	—	45
10	20	10	—	—	—	—	—	—	—	—	−45

Table 10.24: The default hatch patterns used by `HATCH_DRAW`

Re-defining hatch patterns

The hatch patterns can be scaled to fit a user defined world coordinate system by calling the `HATCH_SCALE` routine.

subroutine	`HATCH_SCALE(` $nhatch$, sf `)`		
	variable	*type*	*description*
input	$nhatch$	`INTEGER*4`	pattern number, $1 \leq nhatch \leq 10$
	sf	`REAL*4`	scale factor
output			

sf is a scaling factor for conversion to the user defined world coordinate system of units. For example, if the world coordinate system is 10 units in x by 7.5 units in y, a scale factor $sf = \frac{10}{640}$ might be used. The definition of any one hatch pattern can be changed by calling the `HATCH_SET` routine.

subroutine	`HATCH_SET(` $nhatch$, $space$, $nspace$, $angle$ `)`		
	variable	*type*	*description*
input	$nhatch$	`INTEGER*4`	pattern number, $1 \leq nhatch \leq 10$
	$space$	`REAL*4`	spacing array
	$nspace$	`INTEGER*4`	length of $space$ array
	$angle$	`REAL*4`	hatch line angle (degrees)
output			

The $space$ array should contain the spacings between the lines in the $nhatch$ hatch pattern, and must have no more than 10 elements. The spacings should be in the world coordinate system of units. *nspace* is the number of spacings to use from the *space* array, $1 \leq nspace \leq 10$. The *angle* parameter is the angle of the hatching lines, and is assumed to be in degrees.

Use the HATCH_GET routine to obtain the current definition of a hatch pattern.

subroutine	HATCH_GET(*nhatch*, *space*, *nspace*, *angle*)		
	variable	*type*	*description*
input	*nhatch*	INTEGER*4	pattern number, $1 \leq nhatch \leq 10$
output	*space*	REAL*4	spacing array, length 10
	nspace	INTEGER*4	effective length of *space* array
	angle	REAL*4	hatch line angle (degrees)

The HATCH_GET routine has the same parameters as the HATCH_SET routine, but the current spacings are *returned* in *space*, the number of spacings is *returned* in *npsace*, and the angle is *returned* in *angle*.

10.4 Flushing the plot buffer

Graphics primitives are accumulated in a buffer and are sent to the active monitors in a burst whenever the buffer fills. Two buffers are maintained in memory, one for each monitor. You may find that after your last graphics routine call, to PLOT_R for example, that not all of the picture has been drawn on the monitor. To ensure that all the generated lines and points are sent to the monitor use the FLUSH_PLOT routine.

subroutine	FLUSH_PLOT		
	variable	*type*	*description*
input			
output			

This routine has no affect on a plotter, bitmap or graphical editor file.

10.5 Alphanumeric or transparent mode

To put the first monitor into transparent, or alphanumeric, mode, as opposed to graphics mode, use the TRANSPARENT_MODE routine.

subroutine	TRANSPARENT_MODE(*iclear*)		
	variable	*type*	*description*
input	*iclear*	INTEGER*4	action code
output			

Similarly, to put the second monitor into transparent, or alphanumeric, mode, as opposed to graphics mode, use the `TRANSPARENT2_MODE` routine.

subroutine `TRANSPARENT2_MODE(` $iclear$ `)`

	variable	*type*	*description*
input	$iclear$	`INTEGER*4`	action code
output			

If $iclear \neq 0$ then `CLEAR_PLOT` is called first. Normally, one would call `TRANSPARENT_MODE`, or `TRANSPARENT_MODE`, with $iclear = 0$, and call `CLEAR_PLOT` yourself, if desired.

These routines affect only the appropriate terminal monitor type. Normally, the first monitor is the terminal monitor, so only `TRANSPARENT_MODE` is called.

Placing a terminal monitor into transparent mode is useful for writing normal text output in the middle of a graphics session or to return to non-graphics mode at the end of a graphics session. The graphics image will be preserved and will be unaffected by any text subsequently written to the monitor. Graphics mode is automatically re-entered on the next call to a graphics routine.

Clearing the alphanumeric terminal screen

To clear the non-graphics, or alphanumeric, terminal monitor screen, without affecting the graphics image use the `CLTRANS` routine.

subroutine `CLTRANS`

	variable	*type*	*description*
input			
output			

If the terminal monitor is currently in graphics mode, call `TRANSPARENT_MODE` before calling `CLTRANS`.

10.6 Clearing the graphics

To clear all graphics devices use the `CLEAR_PLOT` routine.

subroutine `CLEAR_PLOT`

	variable	*type*	*description*
input			
output			

This routine clears the graphics terminal monitor and empties the hardcopy bitmap. It also deletes any plot file that is attached to a logical output unit, and then reopens it. `CLEAR_PLOT` is also the plot initialization routine, and, as such, should be called *after* the fundamental control routines have been called to set up your windows and viewports, and *before* any drawing is done.

On a CIT467 or PT100G terminal, or on an X window device, the alphanumeric screen will not be cleared when `CLEAR_PLOT` is called. On all other terminal types, the transparent, or alphanumeric, screen will be cleared along with the graphics screen.

10.7 Selective erasing or complementing

On a terminal monitor, the bitmap, and in PostScript, but *not* on other plotter types, you can draw graphics in three different modes:

- pixels turned on
- pixels turned off
- pixels complemented

Call the `PLOT_DATA_LEVEL` routine to choose which type to use for subsequent graphics.

subroutine	`PLOT_DATA_LEVEL(` *level* `)`		
	variable	*type*	*description*
input	*level*	`INTEGER*4`	action code
output			

The drawing mode is determined by the value of *level*, which is described in Table 10.25 on page 56. The default value of *level* is 0.

ilev	*result*
0	pixels turned on, that is, lit on a terminal monitor and darkened on a bitmap (default)
1	pixels turned off, that is, erased on a terminal and a bitmap
2	pixels complemented, that is, turned on if they're off or turned off if they're on

Table 10.25: Graphics drawing mode

It is important to remember that this has no affect on plot files, that is, once an object is drawn and included into the plot file, it cannot be selectively erased from the file, even if it is selectively erased from the terminal monitor and from the bitmap. The only thing to do is call `CLEAR_PLOT` and redraw the picture.

By switching $level$ to 1 and redrawing a figure, that figure is erased from the terminal monitor and from the bitmap. This will, of course, leave "holes" in lines that the figure overlapped. A useful technique for moving an object on the screen from location to location without leaving holes is to switch to $level = 2$, draw the object at the first location, redraw it at that location, draw it at the next location, redraw it, and so on.

10.8 Graphics cursor input

The graphics cursor, or crosshair, can be used to interactively point to a location on the graphics terminal monitor, to designate the location of a graphical object. Joysticks or light pens can also used be for this purpose. All of the terminal monitor types support such "locator" input. Call the `CROSSHAIR_R` routine to make use of this feature.

subroutine `CROSSHAIR_R(` x, y, $code$, xl, yl `)`

	variable	*type*	*description*
input	xl	`REAL*4`	world coordinate
	yl	`REAL*4`	world coordinate
output	$code$	`LOGICAL*1`	key code
	x	`REAL*4`	world coordinate
	y	`REAL*4`	world coordinate

This routine first activates the graphics cursor and, when possible, pre-positions it at the world coordinates (xl, yl). If (xl, yl) does not map to the terminal monitor screen, the graphics cursor appears where it was last positioned.[7] Once the graphics cursor is positioned, striking any alphanumeric or punctuation key will cause the routine to return with the (x, y) world coordinates of the graphics cursor and the ASCII code of the key in the $code$ variable. If a light pen is used, after positioning the tip of the light pen at any point on the screen and applying enough pressure so that the pen tip retracts into the pen, the (x, y) coordinates and the ASCII code for "@", that is, decimal 64, are returned. If a character is struck prior to triggering the light pen, $code$ will be returned as the ASCII code for that character.

A VT640 display terminal supports either crosshair mode or, optionally, light pen mode. The crosshair is positioned to any point on the screen by using the four arrow keys.[8] A light pen option may be installed on your VT640. To activate the light pen rather than the crosshair, ensure that the `ESC-SUB` parameter in the VT640 Local Mode (reached by striking the `PF3` key

---

[7] The VT640, Regis and Tektronix terminals allow this pre-positioning of the crosshair, but other terminal types do not.

[8] In order for the crosshair routines to work properly, the VT640 "trailer codes" must be set to hexadecimal `0D` and `FF` in local mode.

within five seconds after pressing the `SET-UP RESET` key) is set to zero.

10.9 Colour

Terminal monitor colour, pen number, and bitmap colour can be set with the `PLOT_COLOR` routine.

subroutine	`PLOT_COLOR(` $colr_1$, $colr_2$ `)`		
	variable	*type*	*description*
input	$colr_1$	`INTEGER*4`	first monitor colour code
	$colr_2$	`INTEGER*4`	second monitor colour code
output			

$colr_1$ and $colr_2$ designate the colour codes for monitor 1 and monitor 2, respectively. Refer to Table 10.26 on page 58. If the terminal monitor does not support colour, $colr_1$ is ignored. If the second monitor is off, or does not support colour, $colr_2$ is ignored. If a graphical editor file, that is, an **EDGR** file, has been opened, the colour code $colr_1$ is retained by the editor, but $colr_2$ is not retained.

colour code	X window system	CIT467	TK4107	*pen defaults*
0	black	black	black	—
1	red	red	white	black (thick pen)
2	blue	blue	red	red (thick pen)
3	magenta	magenta	green	green (thick pen)
4	green	green	blue	blue (thick pen)
5	yellow	yellow	cyan	black (thin pen)
6	cyan	cyan	magenta	red (thin pen)
7	white	white	yellow	green (thin pen)
8	coral	—	orange	blue (thin pen)
9	red magenta	—	green yellow	—
10	green cyan	—	green cyan	—
11	blue cyan	—	blue cyan	—
12	—	—	blue magenta	—
13	—	—	red magenta	—
14	—	—	dark grey	—
15	—	—	light grey	—

Table 10.26: Colour codes, terminal colours and pen numbers

A pen plotter is assumed to have four thick pens and four thin pens, hence the duplication of colours. There is no guarantee that a specific pen plotter will have the default pens in the designated positions. It is recommended that the user check the pen plotter for pens and paper before submitting a drawing.

The current colour codes for monitor 1 and monitor 2 can be found in the `PLOT_COLOURS` common block.

```
INTEGER*4 COLR1, COLR2
COMMON /PLOT_COLOURS/ COLR1, COLR2
```

where `COLR1` $= colr_1$, `COLR2` $= colr_2$, as defined in the call to `PLOT_COLOR`.

PostScript

To obtain colour in PostScript drawings, you must use the `POSTSCRIPTCOLOUR` common block.

```
LOGICAL*4 PSCOLOUR
COMMON /POSTSCRIPTCOLOUR/ PSCOLOUR
```

Set the `PSCOLOUR` variable to `.TRUE.` *after* the fundamental control routines have been called to set up your windows and viewports, and *before* any drawing is done. The default value for `PSCOLOUR` is `.FALSE.`

10.10 Symbols

Seven (7) different predefined symbols can be drawn with the `SYMBOL` routine.

subroutine	`SYMBOL(` x, y, $height$, $idum$, $angle$, n `)`		
	variable	*type*	*description*
input	x	`REAL*4`	world coordinate
	y	`REAL*4`	world coordinate
	$height$	`REAL*4`	symbol height in world units
	$idum$	`INTEGER*2`	an unused variable
	$angle$	`REAL*4`	symbol angle
	n	`INTEGER*4`	symbol code
output			

(x, y) is the location, in world coordinates, of the centre of the symbol. $height$ is the height of the symbol, in world units. $idum$ is an unused dummy `INTEGER*2` variable. $angle$ is the angle of the symbol in degrees, measured counterclockwise from the horizontal. The symbol code is given by n, which must be *negative*. For examples of the seven symbols, see Figure 10.13 on page 60.

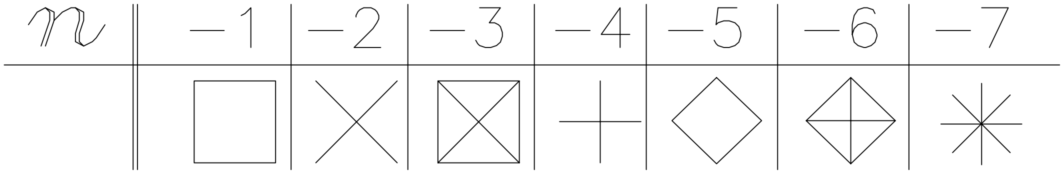


Figure 10.13: The symbols obtainable with the `SYMBOL` routine

10.11 The graphical editor: EDGR

EDGR is a facility for creating, editing, manipulating, and storing graphical data. A document is available, EDGR – GRAPHICS EDITOR USER'S GUIDE, explaining the editor in detail. Only the generation of editor *drawing files* is discussed here.

DWG

In general, one need only call the `DWG` routine.

subroutine `DWG`

	variable	*type*	*description*
input			
output			

The `DWG` routine *interactively* opens and activates a drawing file. When `DWG` is called, the user is prompted at the terminal to enter a name for the new "drawing" to be generated. Two files will be created: drawing_name.`DWG` and drawing_name.`DWT`.

These files form an *interlocked* pair that constitutes the *drawing file* that is refered to in this document.

DWG_BATCH

A non-interactive version is available as the `DWG_BATCH` routine.

subroutine `DWG_BATCH(` $dname$, ier `)`

	variable	*type*	*description*
input	$dname$	`CHARACTER*(*)`	drawing name
output	ier	`INTEGER*4`	error flag

$dname$ is a character variable or literal containing a valid drawing name. ier is returned as zero (0) if the drawing file was successfully opened, and returned as one (1) if the drawing file could not be opened.

DWG_CLOSE

The routine `DWG_CLOSE` is used to close any drawing file that is currently open.

subroutine `DWG_CLOSE`

	variable	*type*	*description*
input			
output			

An open drawing file is closed whenever a drawing file is opened, since there can only be one open drawing file at any time.

Note: `CLEAR_PLOT` does not automatically open, close, or empty any drawing files, in fact, it has no affect on drawing files at all.

DWG_OUTPUT

Use the `DWG_OUTPUT` routine to inhibit the flow of data to an open drawing file.

subroutine `DWG_OUTPUT(` $string$ `)`

	variable	*type*	*description*
input	$string$	`CHARACTER*(*)`	action flag
output			

Call this routine with $string$ = `'OFF'` to disable output to the drawing file, and with $string$ = `'ON'` to enable output to the drawing file.

11 EXAMPLE PROGRAMS

This chapter contains examples of FORTRAN source code that use the graphics routines discussed in the previous chapters.

VMS

```
$FORTRAN PROGRAM.FOR
$LINK PROGRAM.OBJ,GPLOT$DIR:GPLOT/LIB
$RUN PROGRAM.EXE
```

The logical name `GPLOT$DIR` points to the location of the **GPLOT** library, `GPLOT.OLB`.

UNIX

```
%f77 -static -c program.f
%f77 -o program program.o -lgplot -lX11
%program
```

The library `gplot.a` must be installed as `/usr/local/lib/libgplot.a/`. Otherwise, an explicit pathname must be given for `gplot.a`

OSF1

```
%f77 -static -c program.f
%f77 -o program -T 00400000 -D 10000000 program.o -lgplot -lX11
%program
```

The `-T -D` flags force the addresses into the two gigabyte range, which is necessary for 32 bit addressing.

The library `gplot.a` must be installed as `/usr/local/lib/libgplot.a/`
Otherwise, an explicit pathname must be given for `gplot.a`

11.1 Using all the defaults

This example uses all of the default viewing transformations, drawing five nested squares on a VT640 type terminal monitor and generating a bitmap at the same time. The user is queried for hardcopy destinations.

```
CALL CLEAR_PLOT                 ! initialize and clear
SIZE = 300.0
DO I = 1, 5
  CALL PLOT_R(100.,50.,3)      ! pen up to bottom left corner
  CALL PLOT_R(100.,50.+SIZE,2) ! pen down around periphery of square
  CALL PLOT_R(100.+SIZE,50.+SIZE,2)
  CALL PLOT_R(100.+SIZE,50.,2)
  CALL PLOT_R(100.,50.,2)
  SIZE = SIZE * 0.7            ! decrement size of square
END DO
CALL FLUSH_PLOT                 ! flush the graphics to the terminal
CALL NARGSI(1)
CALL GRAPHICS_HARDCOPY(0)       ! inquire about hardcopies
STOP
END
```

An example of the output from this program is shown in Figure 11.14 on page 63.

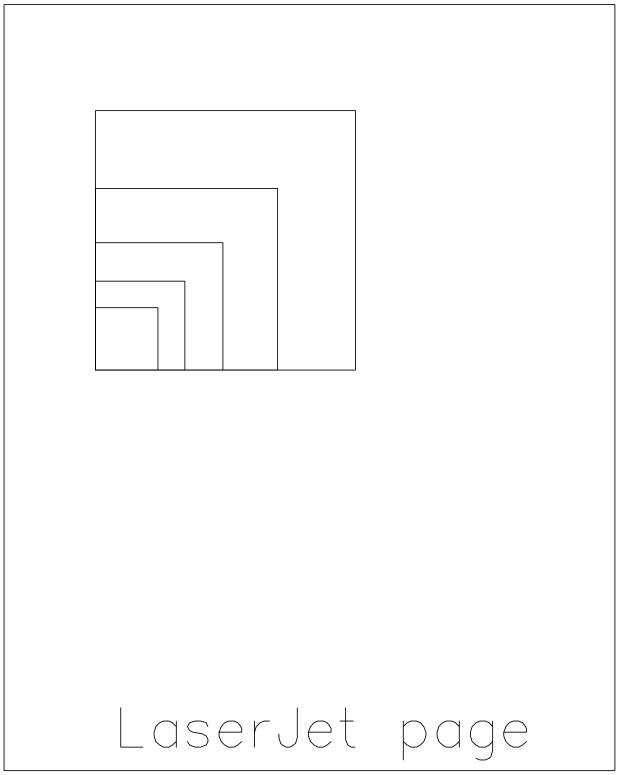


Figure 11.14: Program example using all the defaults

11.2 Using an X window device

Example Programs

This example demonstrates how to generate a graphics editor drawing file while drawing on an X window type terminal. It uses the `SET_PLOT_DEVICES` routine to choose the viewing transformations. Select an X window type monitor and no second monitor, so the world coordinate system is 8.5×11 inches. Draw in inches with a portrait orientation.

Suppose we wish to draw a triangle with the text string “Triangle” at its apex and to have the ability to later edit the drawing.

Note: **EDGR** has the ability to produce hardcopies of the drawing, so there is no need to call `GRAPHICS_HARDCOPY` in this example.

```
CALL SET_PLOT_DEVICES(18,6,0,7,2,'IN','PORTRAIT',1)
CALL CLEAR_PLOT                        ! initialize and clear
CALL DWG_BATCH('EXAMPLE2',IERR)        ! open EDGR drawing file
IF( IERR .EQ. 1 )STOP 'Unable to open drawing file'
CALL PLOT_COLOR(2,1)                   ! draw the triangle in blue
CALL PLOT_R(1.,1.,3)                   ! pen up to bottom left corner
CALL PLOT_R(4.,8.,2)                   ! pen down to apex
CALL PLOT_R(7.,1.,2)                   ! pen down to bottom right corner
CALL PLOT_R(1.,1.,2)                   ! connect to starting point
CALL PLOT_COLOR(4,2)                   ! draw text in green
CALL PSYM(4.1,8.1,.5,'Triangle',0.,8) ! text string 1/2 inch high
STOP
END
```

An example of the output from this program is shown in Figure 11.15 on page 65. This program should be compiled, linked, and run as in the previous example.

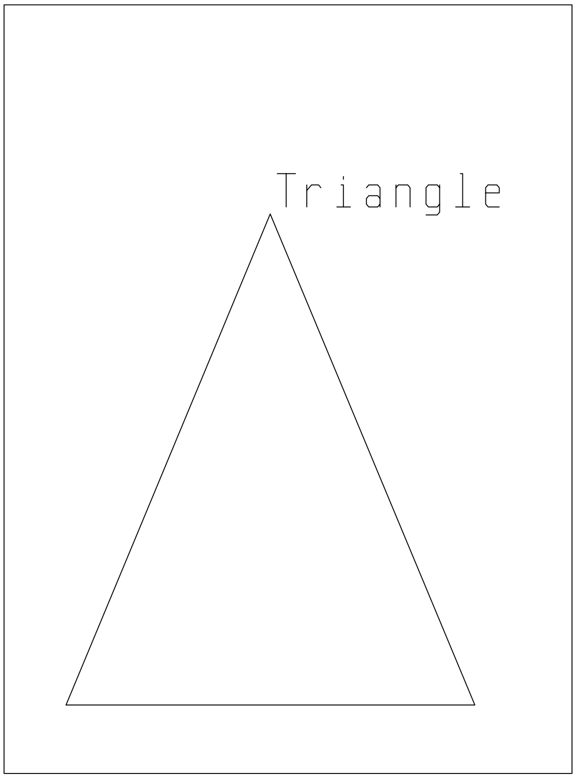


Figure 11.15: Program example using X window device

12 GRAPH PLOTTING

There are several routines in the graph plotting package. These are listed in Table 12.27.

GPLOT_SETUP	determines your terminal type and desired plotter type, defines a useful set of viewing transformations and, optionally, restores a set of graph characteristic values from a file
GPLOTI	initializes all of the graph characteristic values to their default values
GPLOT	draws axes and/or plots data, error bars, histograms, etc.
GPLOT_R	line drawing routine similar to PLOT_R, except the (x, y) location is given in graph units instead of world units
GPLOT_CONVERT	converts (x, y) locations from graph units to world units, or vise versa
GETNAM	a function that returns the current value of a numeric graph characteristic
SETNAM	sets the graph characteristic values which are numeric
GETLAB	a function that returns the current values of graph characteristics which are text strings
SETLAB	sets the graph characteristics which are text strings
GPLOT_SAVE_FILE	saves the current set of graph characteristics in an editable file
GPLOT_RESTORE_FILE	restores a file of graph characteristics
GPLOT_CONTROL	allows the user to interactively control a drawing by entering commands

Table 12.27: Graph plotting routines

12.1 GPLOT_SETUP

The GPLOT_SETUP routine allows you to define a set of default graph characteristics different than the standard defaults, and to select a pre-defined set of viewing transformations.

subroutine GPLOT_SETUP($filename$)

	variable	*type*	*description*
input	$filename$	CHARACTER*(*)	
output			

$filename$ is the name of a file which will be passed on to the routine GPLOT_RESTORE_FILE. If $filename$ is blank, no file will be restored, so the graph characteristics will be set to their standard default values. If the file cannot be opened, a message to that effect will be displayed, and the graph characteristics will be set to their standard default values.

The GPLOT_SETUP routine is a convenient way to select a pre-defined set of viewing transformations. The device configuration routine SET_PLOT_DEVICES is called with the values as shown in Table 12.28.

u_1	the screen output unit is set to 6
u_2	the plotter output unit is set to 7
$ibit$	the bitmap size code is set to 0
$units$	the plotting units code is set to 'CM'
$mode$	the orientation code is set to 'LANDSCAPE'
np	the number of pages is set to 1

Table 12.28: Device configuration with GPLOT_SETUP routine

To determine the terminal monitor code, t_1, the logical name under VMS, or environment variable under UNIX, TRIUMF_TERMINAL_TYPE is translated, and if it has been assigned a value, this value will be used for the terminal type. If TRIUMF_TERMINAL_TYPE is unassigned, a list of valid terminal types is displayed and the user will be asked to interactively enter the terminal and plotter types. t_2, is determined by translating the logical name or environment variable, TRIUMF_PLOTTER_TYPE and choosing the appropriate value for the plotter code. If TRIUMF_PLOTTER_TYPE is unassigned, an HP plotter type, size A, is assumed.[9] The world coordinate system units ranges that will be in effect are shown in Table 9.17 on page 41.

12.2 GPLOTI

The GPLOTI subroutine is used to initialize, or re-set, all of the graph keywords and labels to their default values. It has no parameters.

---

[9] The terminal type must be known, but the plotter type can be defaulted.

subroutine	GPLOTI		
	variable	*type*	*description*
input			
output			

Important: If this routine is not called explicitly in the user's program, it will be automatically called, once, by the first **Graph Plotting** routine, that is, one of the routines listed in Table 12.27 on page 66, which is called in the user's program. This can cause unexpected results. For example, if you do not call `GPLOTI` but set the colour using a call to the `PLOT_COLOR` routine, and then call `GPLOT` to plot a graph, the `GPLOT` routine will call `GPLOTI` for you, which will reset the colour back to the default. It is a good idea to initialize the graph plotting routines, with a call to `GPLOTI`, right after you initialize the low level graphics routines, with a call to `CLEAR_PLOT`.

12.3 GPLOT

The `GPLOT` routine draws axes and plots data. It also is capable of drawing error bars, histograms and filling the area under curves. `GPLOT` is called with eight parameters, where the last four are optional.

subroutine	GPLOT(x, y, npt, $iaxis$, $pchar$, $psize$, $pcolr$, $pangl$)		
	variable	*type*	*description*
input	x	REAL*4	data coordinate array (independent)
	y	REAL*4	data coordinate array (dependent)
	npt	INTEGER*4	number of data points
	$iaxis$	INTEGER*4	action code
	$pchar$	LOGICAL*1	plotting symbol code array
	$psize$	REAL*4	plotting symbol size array
	$pcolr$	LOGICAL*1	plotting symbol colour array
	$pangl$	REAL*4	plotting symbol angle array
output			

The `GPLOT` routine requires a previous call to the `NARGSI` routine.

Example:

```
...
CALL NARGSI(4)
CALL GPLOT(X,Y,N,1)
...
CALL NARGSI(8)
CALL GPLOT(X,Y,N,1,P,S,C,A)
...
```

The `NARGSI` routine informs the `GPLOT` routine of the actual number of arguments, $nargs$, that the `GPLOT` routine has been passed. This somewhat confusing arrangement is required since the `GPLOT` routine can be called with a variable number of arguments. Under VMS, a method was developed whereby the actual number of arguments in a subroutine call could be determined at run time. The UNIX/RISC architecture uses a different argument passing mechanism from VMS. Routines with a variable number of arguments now require the number of actual arguments to be passed via a call to `NARGSI`. This does not apply to alternate return line numbers, which are optional in the argument list, but are not arguments in the usual sense.

Required arguments

x	an array dimensioned npt or npt_1, containing the independent variable data. If no horizontal error bars are desired, then x is assumed to be singly dimensioned. If horizontal error bars are desired, then x is assumed to have two dimensions. See the description of `ERRBAR`, later on in this document, for more information on error bars.
y	an array dimensioned npt or npt_1, containing the dependent variable data. If no vertical error bars are desired, then y is assumed to be singly dimensioned. If vertical error bars are desired, then y is assumed to have two dimensions.
npt	a number, or an array dimensioned 2, containing the number of data points to be plotted. If no error bars are desired, then npt is assumed to be a single number, but if horizontal and/or vertical error bars are desired, then npt is assumed to be an array with length two (2), where npt_1 is the number of data points to plot, and npt_2 is the actual first dimension of the arrays x and/or y as declared in the calling program. npt, or npt_1, must be > 0.
$iaxis$	a code number with the following affect:

	result
$iaxis = 1$	the axes are drawn first and then the data points
$iaxis = 2$	only the data are plotted, with no axes
$iaxis = 3$	the data are plotted first and then the axes
$iaxis = 4$	only the axes are drawn, with no data

Optional arguments

The following `GPLOT` arguments are optional and their interpretation is dependent on the

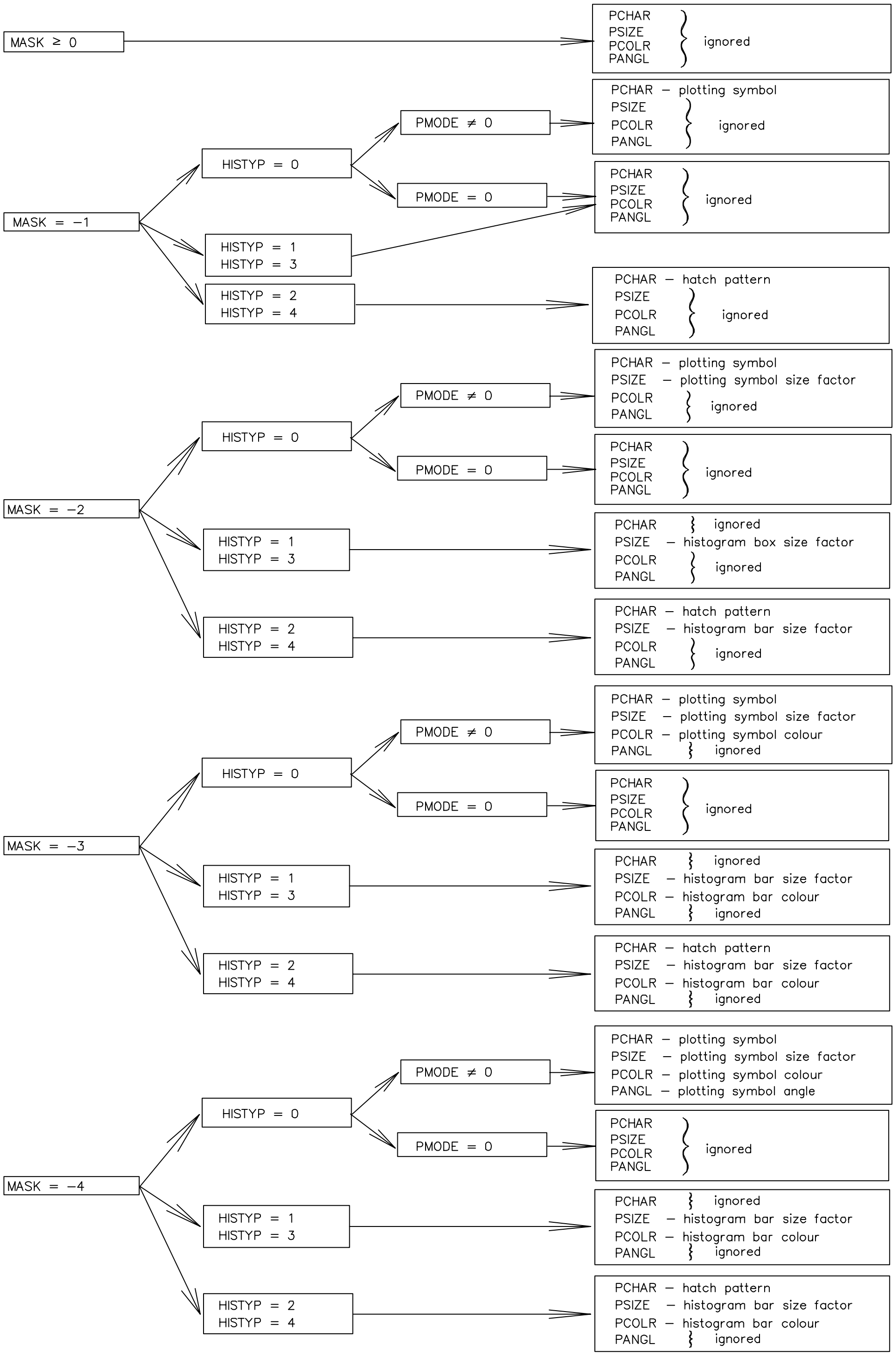


Figure 12.16: Optional `GPLOT` arguments

values of the graph keywords `MASK`, `HISTYP` and `PMODE`. Figure 12.16 on page 70 indicates how the optional arguments are used.

$pchar$	an array dimensioned npt or npt_1, interpreted either as a decimal number corresponding to a symbol or as the hatch pattern to fill a histogram bar. Refer to Figure 12.17 on page 72. When drawing a plotting symbol, if `PMODE` $\times\ pchar_i > 0$ then the plotting symbol at (x_i, y_i) will be connected by a line segment to (x_{i-1}, y_{i-1}), if `PMODE` $\times\ pchar_i < 0$ then the plotting symbol at (x_i, y_i) will be disconnected from (x_{i-1}, y_{i-1}). When filling histogram bars, if $pchar_i > 0$ then the outline of the bar at (x_i, y_i) is drawn and filled, if $pchar_i < 0$ then the outline of the bar at (x_i, y_i) is not drawn but the filling is done.
$psize$	an array dimensioned npt or npt_1, is an optional argument *only if* the $pchar$ argument is absent. $psize_i$ is interpreted either as the relative size of the plotting symbol to draw at (x_i, y_i); or as the relative size of the histogram bar at (x_i, y_i). The plotting symbol size will be $pchar_i \times$ `CHARSZ`. The sizes of the histogram bars are described in Table 12.29 on pageref 72.
$pcolr$	an array dimensioned npt or npt_1, is an optional argument *only if* the $psize$ argument is absent. $pcolr_i$ is interpreted either as the colour of the plotting symbol to be drawn at (x_i, y_i); or as the colour of the histogram bar at (x_i, y_i). To change the colour of the line segment between data points without drawing a plotting symbol, use $pchar_i = 0$.
$pangl$	an array dimensioned npt or npt_1, is an optional argument *only if* the $pcolr$ argument is absent. $pangl_i$ is interpreted as the angle, in degrees, of the plotting symbol to draw at (x_i, y_i). $pangl$ is ignored if `HISTYP` is not zero.

12.3.0.1 Colour

Table 12.30 on page 73 details how the colour code number is mapped to the different colours.

Note: The codes are different than those listed in the section describing the call to `PLOT_COLOR`, as listed in Table 10.26 on page 58.

Some devices, such as the CIT467 terminal, only support eight colours, including black (0). A colour code > 7 on such a device, will be interpreted as white (7). There is no guarantee that the correct pen will be in the correct position on a pen plotter, so the pen positions should be confirmed before submitting a plot file to a pen plotter.

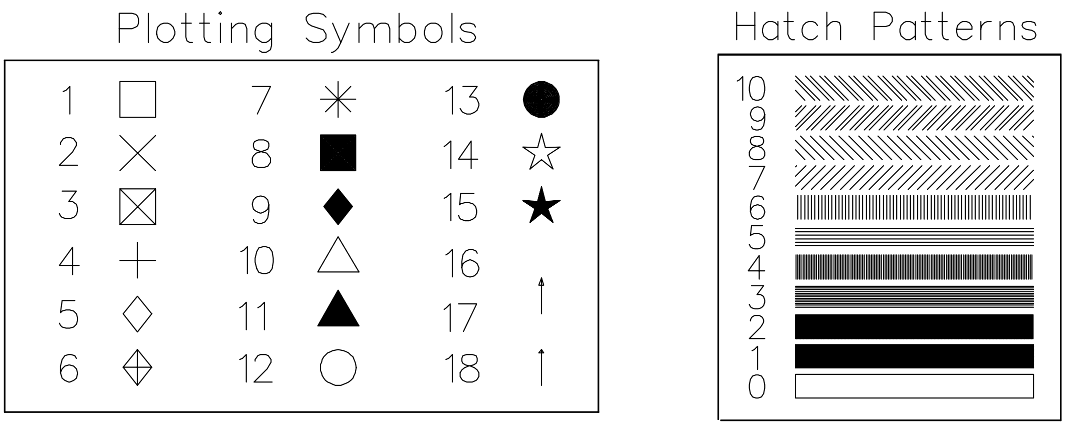


Figure 12.17: Plotting symbols and hatch patterns

HISTYP	*plotting symbol at* (x_i, y_i)	
0	for $1 \leq i \leq npt$	size $= psize_i \times$ CHARSZ$_i$

HISTYP	*vertical histogram bar at* (x_i, y_i)	
1 or 2	for $i = 1$	width $= psize_i \times (x_2 - x_1)$
	for $i < 1 < npt$	width $= psize_i \times (x_{i+1} - 2 \times x_i + x_{i-1})/2$
	for $i = npt$	width $= psize_i \times (x_{npt} - x_{npt-1})$

HISTYP	*horizontal histogram bar at* (x_i, y_i)	
3 or 4	for $i = 1$	height $= psize_i \times (y_2 - y_1)$
	for $i < 1 < npt$	height $= psize_i \times (y_{i+1} - 2 \times y_i + y_{i-1})/2$
	for $i = npt$	height $= psize_i \times (y_{npt} - y_{npt-1})$

Table 12.29: The relationship of `HISTYP` to plotting symbol size and histogram bar width

$pcolr_i$	*colour*	*plotter pen*
0	black	—
1	white	black (thick pen)
2	red	red (thick pen)
3	green	green (thick pen)
4	blue	blue (thick pen)
5	yellow	black (thin pen)
6	cyan	red (thin pen)
7	magenta	green (thin pen)
8	coral	blue (thin pen)
9	red magenta	—
10	green cyan	—
11	blue cyan	—

Table 12.30: `GPLOT` Colours

12.4 GPLOT_R

This routine draws straight line segments, similar to the `PLOT_R` routine.

subroutine `GPLOT_R(` x, y, $ipen$ `)`

	variable	*type*	*description*
input	x	`REAL*4`	data coordinate
	y	`REAL*4`	data coordinate
	$ipen$	`INTEGER*4`	virtual pen code
output	x	`REAL*4`	converted to world coordinates if $ipen = -1$
	y	`REAL*4`	converted to world coordinates if $ipen = -1$

x and y are the graph coordinates to which the virtual pen will move, and $ipen$ is the code that controls the virtual pen movement. Refer to Table 12.31 on page 73.

$ipen$	*result*
-1	convert (x, y) to world coordinates, do not draw anything
2	draw from the current position to (x, y)
-2	draw from the current position to (x, y), end a dashed line sequence
3	move to (x, y) without drawing

Table 12.31: Pen code for routine `GPLOT_R`

The line type used will be the current value of the `LINTYP` keyword. The line segments will be clipped at the boundaries of the graph box. When drawing dashed lines, you should draw to the last location with $ipen = -2$ instead of $+2$. This ends the dashing sequence by extending the last dash to the end of the line.

12.5 GPLOT_CONVERT

This subroutine converts from graph units to world units or from world units to graph units.

subroutine `GPLOT_CONVERT(` x, y, u, v, $code$ `)`

	variable	*type*	*description*
input	x	`REAL*4`	input coordinate
	y	`REAL*4`	input coordinate
	$code$	`INTEGER*4`	conversion code
output	u	`REAL*4`	output coordinate
	v	`REAL*4`	output coordinate

The conversion depends on the value of $code$, as shown in Table 12.32 on page 74.

$code$	*result*
$+1$	convert from graph units to world units and get the current plot keyword values needed for the conversion
-1	convert from graph units to world units and use the previously obtained plot keyword values
$+2$	convert from world units to graph units and get the current plot keyword values needed for the conversion
-2	convert from world units to graph units and use the previously obtained plot keyword values

Table 12.32: Conversion code for `GPLOT_CONVERT`

12.6 GETNAM

The `GETNAM` function returns the current value of a graph keyword.

`REAL*4` *function* `GETNAM(` $name$ `)`

	variable	*type*	*description*
arguments	$name$	`CHARACTER*(*)`	keyword name

`GETNAM` gets the percentage value, if applicable, of the keyword if $name(1:1) =$ ‘`%`’.

Example:

```
X  = GETNAM('XMIN')
YP = GETNAM('%YLAXIS')
```

will set `X` to the current minimum value on the x-axis, and will set `YP` to the current location of the lower end of the y-axis as a percentage of the height of the window.

If $name$ is not a valid keyword name, `GETNAM` returns the value 9.99×10^{-37} and writes on unit number `IOUTS` the following message:

```
*** GETNAM ERROR: No such GPLOT name
```

The unit number `IOUTS` is defined in the `PLOT_OUTPUT_UNIT` common block.

```
INTEGER*4 IOUTS
COMMON /PLOT_OUTPUT_UNIT/ IOUTS
```

`IOUTS` is the standard output unit for messages written from the graphics subroutines. The default value of `IOUTS` is 6.

If $name$ = '`MENU`', then a list of most of the plot specification names and their current values is written on unit `IOUTS`, and a value of zero is returned. See Table 12.33 on page 76 for an example of this menu. In this menu, percentage values are given for applicable names. This menu is easier to read if the user's terminal has been previously set to a width of 132.[10]

If $name$ = '`MENUS`', a short list of keywords and their current values is displayed, see Table 12.34 on page 77. This short list contains the names that are most often changed by the user.

If $name$ = '`SPECIAL`', then a chart of the reserved character names will be drawn on your monitor screen. See Figure 13.19 on page 90. These names are the special characters that can be drawn with the `SETLAB` routine using the keyword `TEXT`. The chart will be drawn in graphics, but will not be entered into any plot files or any open **EDGR** drawing file.

12.7 SETNAM

The `SETNAM` subroutine sets a keyword to a specified value.

---

[10] Under VMS: To set your terminal to a width of 132 use the DCL command: `$ SET TERMINAL/WIDTH=132`

MASK	=	0.000		BOX	=	1.000		CHARSZ	=	4.790	1.000%	CHARA	=	0.000	0.000%
PMODE	=	1.000		ALIAS	=	1.000		HISTYP	=	0.000		LINTYP	=	1.000	
PTYPE	=	0.000		COLOUR	=	1.000		CURSOR	=	1.000		ERRBAR	=	0.000	
XLWIND	=	0.000	0.000%	XUWIND	=	639.000	100.000%	XLAXIS	=	76.680	12.000%	XUAXIS	=	600.660	94.000%
NXDIG	=	5.000		NXDEC	=	-1.000		XPOW	=	0.000		XPAUTO	=	1.000	
XNUMSZ	=	9.580	2.000%	XLABSZ	=	14.370	3.000%	XTICL	=	9.580	2.000%	XTICS	=	4.790	1.000%
XTICA	=	270.000	-90.000%	XCROSS	=	0.000		XMIN	=	0.000	0.000	XMAX	=	10.000	10.000
NLXINC	=	5.000		NSXINC	=	5.000		XAUTO	=	2.000		XITICL	=	14.370	3.000%
XITICA	=	270.000	-90.000%	XNUMA	=	0.000	0.000%	XTICTP	=	1.000		BOTTIC	=	1.000	
BOTNUM	=	0.000		TOPTIC	=	-1.000		TOPNUM	=	0.000		NXGRID	=	0.000	
XAXIS	=	1.000		XAXISA	=	0.000		XLOG	=	0.000		XZERO	=	0.000	
YLWIND	=	0.000	0.000%	YUWIND	=	479.000	100.000%	YLAXIS	=	57.480	12.000%	YUAXIS	=	450.260	94.000%
NYDIG	=	5.000		NYDEC	=	-1.000		YPOW	=	0.000		YPAUTO	=	1.000	
YNUMSZ	=	9.580	2.000%	YLABSZ	=	14.370	3.000%	YTICL	=	9.580	2.000%	YTICS	=	4.790	1.000%
YTICA	=	90.000	90.000%	YCROSS	=	0.000		YMIN	=	0.000	0.000	YMAX	=	10.000	10.000
NLYINC	=	5.000		NSYINC	=	5.000		YAUTO	=	2.000		YITICL	=	14.370	3.000%
YITICA	=	90.000	90.000%	YNUMA	=	-90.000	-90.000%	YTICTP	=	1.000		LEFTIC	=	1.000	
LEFNUM	=	0.000		RITTIC	=	-1.000		RITNUM	=	0.000		NYGRID	=	0.000	
YAXIS	=	1.000		YAXISA	=	90.000		YLOG	=	0.000		YZERO	=	0.000	
TXTHIT	=	14.370	3.000%	TXTANG	=	0.000		XLOC	=	319.500	50.000%	YLOC	=	239.500	50.000%
XLABEL	=														
YLABEL	=														

Table 12.33: The full menu of GPLOT keywords, with values in centimeters

```
+----------------------------------+----------------------------------+
| MASK     =     0.000             | CHARSZ   =   4.790        1.000% |
| PMODE    =     1.000             | HISTYP   =   0.000               |
| LINTYP   =     1.000 | LINTHK   =     1.000 | COLOUR   =     1.000  |
| XLWIND   =     0.000      0.000% | XUWIND   =  639.000     100.000% |
| XLAXIS   =    76.680     12.000% | XUAXIS   =  600.660      94.000% |
| NXDIG    =     5.000             | NXDEC    =   -1.000              |
| XPOW     =     0.000             | XPAUTO   =    1.000              |
| XMIN     =     0.000       0.000  | XMAX     =   10.000      10.000  |
| NLXINC   =     5.000             | NSXINC   =    5.000              |
| XAUTO    =     2.000             | XLOG     =    0.000              |
| YLWIND   =     0.000      0.000% | YUWIND   =  479.000     100.000% |
| YLAXIS   =    57.480     12.000% | YUAXIS   =  450.260      94.000% |
| NYDIG    =     5.000             | NYDEC    =   -1.000              |
| YPOW     =     0.000             | YPAUTO   =    1.000              |
| YMIN     =     0.000       0.000  | YMAX     =   10.000      10.000  |
| NLYINC   =     5.000             | NSYINC   =    5.000              |
| YAUTO    =     2.000             | YLOG     =    0.000              |
| XLOC     =   319.500     50.000% | YLOC     =  239.500      50.000% |
+----------------------------------+----------------------------------+
XLABEL =
YLABEL =
+----------------------------------+----------------------------------+
```

Table 12.34: The short menu of GPLOT keywords, with values in centimeters

subroutine `SETNAM(` $name$, $value$, * `)`

	variable	*type*	*description*
input	$name$	`CHARACTER*(*)`	keyword name
	$value$	`REAL*4`	keyword value
output			

Set a keyword to a percentage value by having $name(1:1) =$ `'%'`. If $name$ is not a valid keyword, the alternate return is taken, and the message

```
*** SETNAM ERROR: No such GPLOT name
```

is displayed on unit `IOUTS`. In this case, no **GPLOT** common block values are changed.

For example:

```
CALL SETNAM('XMIN',X)
```

will set the minimum value on the x-axis to the current value of `X`.

12.8 GETLAB

This routine is a `CHARACTER*(*)` function that returns the current value of a plot label.

`CHARACTER*(*)` *function* `GETLAB(` $name$ `)`

	variable	*type*	*description*
arguments	$name$	`CHARACTER*(*)`	keyword name

If $name$ is not a valid keyword, `GETLAB` will be returned as a blank string, and the message

```
*** GETLAB ERROR: No such GPLOT name
```

will be written on unit `IOUTS`.

Example:

```
      CHARACTER*132 STRING, GETLAB
      STRING = GETLAB('XLABEL')
```

will set `STRING` to the current value of the x-axis automatic label.

12.9 SETLAB

This routine sets the value of a character valued keyword.

subroutine SETLAB(*name*, *string*, *)

	variable	*type*	*description*
input	*name*	CHARACTER*(*)	keyword name
	string	CHARACTER*(*)	keyword value
output			

The maximum number of characters of *string* that will be stored is 255. If *name* is not a valid keyword, the alternate return is taken, no keyword values are changed, and the message

```
*** SETLAB ERROR: is not a GPLOT name
```

will be written on unit IOUTS.

For example:

```
CALL SETLAB('XLABEL','This is the x-axis label')
```

will set the x-axis automatic label to the specified character string.

12.10 GPLOT_SAVE_FILE

This routine saves the current set of plot characteristics in an editable ASCII file.

subroutine GPLOT_SAVE_FILE(*fname*)

	variable	*type*	*description*
input	*fname*	CHARACTER*(*)	file name
output			

All of the **GPLOT** keywords and their current values will be written to the file, *fname*, with the character valued keywords following the numeric keywords. Thus, one can save all the characteristics that are necessary to reproduce a specific **GPLOT** environment. These characteristics can be restored with the routine GPLOT_RESTORE_FILE.

12.11 GPLOT_RESTORE_FILE

This subroutine restores the set of keywords that are read from a file.

subroutine GPLOT_RESTORE_FILE(*fname*)

	variable	*type*	*description*
input	*fname*	CHARACTER*(*)	file name
output			

The file, *fname*, should contain **GPLOT** keywords and values. Character valued keywords can also be restored. Thus, one can restore all the characteristics that are necessary to reproduce a specific **GPLOT** environment, although it is not necessary to have all **GPLOT** keywords included in the file. These characteristics can be produced with the routine `GPLOT_SAVE_FILE`, or the file may be produced yourself. The format that is expected is a keyword followed by a value. This name and value should be seperated by one or more blanks. The value can be in any format. For character valued keywords, the keyword should be followed by the string value *enclosed in double quotes*. Any line in the file that starts with a ! is considered a comment line, and is ignored. For example, the following is a valid portion of a **GPLOT** restore file:

```
XMIN   10
XMAX 20.0
YMIN 1.E-2
YMAX .09
! The following are plot labels
XLABEL "<theta,_,minus>degrees"
YLABEL "<oint>xdx"
```

12.12 GPLOT_CONTROL

This routine allows the programmer, with a minimum of effort, to turn his/her program into a command driven program giving the user some control over the drawing.

subroutine	`GPLOT_CONTROL(` *prompt* `)`		
	variable	*type*	*description*
input	*prompt*	`CHARACTER*(*)`	keyboard input prompt
output			

The prompt will be displayed at the bottom of the monitor screen and indicates that you are to enter some command, or simply type the `<RETURN>` key. If the `<RETURN>` key is typed without entering anything, program control is passed to the next line in your program. If the `RETURN` command is entered at the prompt, program control is passed to the alternate return statement label that is provided in the call to the `GPLOT_CONTROL` subroutine.

The `GPLOT_CONTROL` commands are listed in Table 12.35 on page 81 and in Table 12.36 on page 82.

`EDGRAPH`	invokes the graphics editor, **EDGR**. If `OPEN` is entered, a drawing file will be opened. The file name can be entered directly, or it will be requested by **EDGR**. If `CLOSE` is entered, the current drawing file will be closed. No subsequent graphics will be entered into this drawing file.
`SET`	allows the user to change the value of any of the **GPLOT** keywords. If no keyword is entered, the user will be asked to enter a keyword and a value. The user will then be asked for another keyword and value, and so on. To terminate this input, simply type the `<RETURN>` key without entering a keyword. If a keyword is entered with the `SET` command, but not a value, then the current value of that keyword will be displayed and a new value may be entered. If no value is entered, the value is left unchanged. The user will only be asked to change the value for this one keyword. If a keyword and a value are entered with the `SET` command, that keyword's value will be changed to the new value. Nothing will be displayed and the user is returned to the prompt.
`MENU`	displays the menu of commands.
`SMENU`	displays the short menu of **GPLOT** keywords.
`LMENU`	displays the complete menu of **GPLOT** keywords.
`SYS`	allows the user to issue an operating system command, e.g., edit a file.
`SAVE`	saves the current state of the **GPLOT** keywords, using the `GPLOT_SAVE_FILE` routine.
`RESTORE`	restores a file of **GPLOT** keywords, using the `GPLOT_RESTORE_FILE` routine.
`HARDCOPY`	inquires about graphics hardcopies, using the `GRAPHICS_HARDCOPY` routine.

Table 12.35: The `GPLOT_CONTROL` menu (part 1)

CLEAR	clears the graphics, using the CLEAR_PLOT routine.
ACLEAR	clears the alphanumeric monitor screen, using the CLTRANS routine. This has no affect on the graphics.
TEXT	allows the user to draw text strings on the plot. You have access to all the text formatting commands and the special reserved characters. This command makes use of the SETLAB routine.
XLABEL	sets the automatic x-axis text label, using the SETLAB routine.
YLABEL	sets the automatic y-axis text label,, using the SETLAB routine.
PCHAR	allows the user to set the plotting symbol that will be drawn at the data points in future graphs. Refer to the description of CHAR under the **Plot labels** section.
GPLOTI	resets all the **GPLOT** keywords to their default values, by calling the GPLOTI routine.
FONT	allows the user to change the text font. If no font name is entered, a list of valid font names is displayed.
RETURN	causes program execution to proceed to the alternate return as specified by the statement label passed in the call to GPLOT_CONTROL.

Table 12.36: The GPLOT_CONTROL menu (part 2)

Following is a simple example illustrating the use of `GPLOT_SETUP` and `GPLOT_CONTROL`.

```
      REAL*4  X(30), Y(30)
      DO I = 1, 30
        X(I) = (I-1.)*180./29.
        Y(I) = SIND(X(I))
      END DO
      CALL GPLOT_SETUP(' ')
      CALL CLEAR_PLOT
10    CALL NARGSI(4)
      CALL GPLOT(X,Y,30,1)
      CALL GPLOT_CONTROL('Plot control >>',&10)
      STOP 'Finished...'
      END
```

13 SETLAB/GETLAB KEYWORDS

The keywords refered to in this chapter are the keywords that can be passed to the `GETLAB` and `SETLAB` routines. These are the entities that control the automatic axis text labels, the text strings, and the data plotting symbols. The following is a list of the keywords, with a very terse description.

13.1 Summary

Following is a list of the keywords, with very terse descriptions.

Text keywords

name	*description*
`TEXT`	draw a text string
`FONT`	change the font

Plotting symbol keywords

name	*description*
`CHAR`	set the data plotting symbol(s)
`HEXCHR`	set the data plotting symbol(s) in hexadecimal code

Axis box keywords

name	*description*
`XLABEL`	automatic x-axis text label
`YLABEL`	automatic y-axis text label

13.2 Text keywords

TEXT

Default value: `TEXT = ' '` (blank)

The `TEXT` keyword allows the user to draw text strings. The text string passed to `SETLAB` can have a maximum length of 255 characters. The text string can have embedded formatting commands which allow for font, height, colour, and other changes; as well as certain pre-defined special characters, such as Greek letters and some mathematical symbols. See the **Text formatting** section, page 89, for information on these formatting commands.

If `CURSOR` is greater than zero, the graphics cursor will be used to locate the reference position for the text string, and the text justification will be determined interactively. If `CURSOR` is less than or equal to zero, the values of `XLOC` and `YLOC` will be used to locate this reference position and the text justification will be determined by the value of `CURSOR`. The height and angle of the text will be `TXTHIT` and `TXTANG`. See the complete descriptions of all these keywords elsewhere in this document.

FONT

Default value: `FONT = STANDARD`

This keyword allows the user to change the text font. This font will be used for drawing text using the `TEXT` keyword, but is also used for the automatic axis labels `XLABEL` and `YLABEL`, and the numbers that label the axes on a graph.

For example:

```
   CALL SETLAB( 'FONT', 'TRIUMF.2' )
```

will change the text font to `TRIUMF.2`. This font will then be used for all text plotting including the numbering on the axes and any axis labels.

The font can also be changed with the `PFONT` subroutine, for example:

```
   CALL PFONT( STRING, 0 )
```

where `STRING` is a valid font name. The currently available fonts are listed in Table 10.22 on page 48.

13.3 Plotting symbol keywords

CHAR

Default value: `CHAR(1:1)` is a ‘box’, `CHAR(2:2)` is a ‘cross’

`CHAR` is one way to control which plotting characters to draw at the data points on a graph. Other ways to set the plotting symbols are the keyword `HEXCHR`; and *pchar*, which is one of the optional arguments for the `GPLOT` routine. When label `CHAR` is set, the `HEXCHR` label is set to the corresponding characters. The keyword `MASK` is associated with `CHAR`, as is shown in the following table.

`MASK`	*result*
< 1	`CHAR` will not be used
$= 1$	`CHAR(1:1)` will be drawn at each data point
> 1	`CHAR(2:2)` will be drawn at every $\texttt{MASK}^{th}$ data point and `CHAR(1:1)` will be drawn at the other points

Furthermore, the keyword `PMODE` controls whether or not the data points will be connected with line segments. If $\texttt{PMODE} = +1$, they will be connected; while if $\texttt{PMODE} = -1$, they will not be connected. If $\texttt{PMODE} = 0$, then `MASK` is ignored, no plotting symbol is drawn, and the data points are connected with line segments.

`CHAR` should be used to draw keyboard characters at the data points, for example:

```
CALL SETLAB( 'CHAR', 'AB' )
```

This will set `CHAR(1:1)` to `A` and `CHAR(2:2)` to `B`. The keyboard character symbols will not be centred at the data points, but will be drawn with the lower left corner at the data point. To use the label `CHAR` to set the special plotting symbols, use the intrinsic FORTRAN function, `CHAR`. For example:

```
CALL SETLAB( 'CHAR', CHAR(1)//CHAR(2) )
```

This will set `CHAR(1:1)` to a ‘box’ and `CHAR(2:2)` to a ‘cross’. The special symbols are centred at the data points.

HEXCHR

Default value: `HEXCHR(1:2) = 01`, `HEXCHR(3:4) = 02`

`HEXCHR` controls which plotting symbols to draw at the data points on a graph. Other ways to set the plotting characters are: the `CHAR` keyword; and $pchar$, which is an optional argument for the `GPLOT` routine. `HEXCHR` acts the same as `CHAR`, but it expects a character string that will be interpreted as a set of *hexadecimal* digit pairs. When keyword `HEXCHR` is set, the `CHAR` keyword is set to the corresponding characters. The keyword `MASK` is associated with `HEXCHR`, as is shown in the following table.

MASK	*result*
< 1	HEXCHR will not be used
$= 1$	HEXCHR(1:2) will be drawn at each data point
> 1	HEXCHR(3:4) will be drawn at every MASKth data point and HEXCHR(1:2) will be drawn at the other points

Furthermore, the plot characteristic PMODE controls whether or not the data points will be connected with line segments. If PMODE $= +1$, they will be connected; while if PMODE $= -1$, they will not be connected. If PMODE $= 0$, then MASK is ignored, no plotting symbol is drawn, and the data points are connected with line segments.

HEXCHR should be used to draw the special symbols centred at the data points, for example:

```
CALL SETLAB( 'HEXCHR', '010C' )
```

This will set HEXCHR(1:2) to a 'box' and HEXCHR(3:4) to a 'circle'. See Figure 13.18 for examples of the special plotting symbols with their hexedical codes. The special symbols are always centred at the data points.

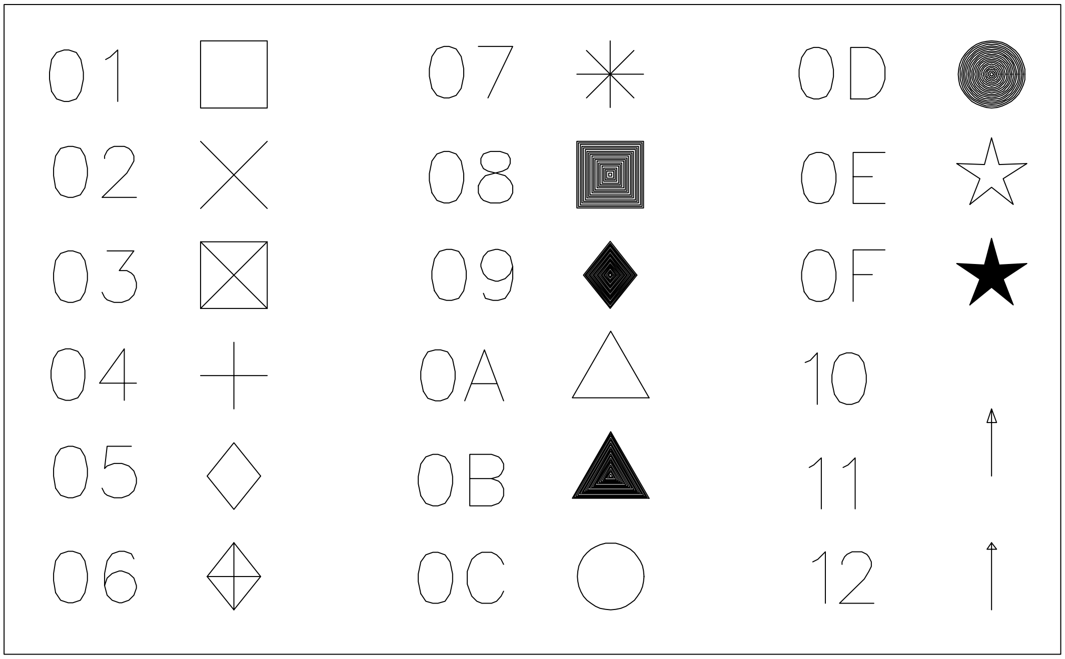


Figure 13.18: Special plotting symbols and hexadecimal codes

13.4 Axis box keywords

SETLAB/GETLAB keywords

XLABEL

Default value: `XLABEL = ' '` (blank)

`XLABEL` controls the x-axis automatic text label. The maximum length of `XLABEL` is 255 characters. The complete x-axis text label is composed of two parts:

- the string defined by `XLABEL`, and
- the scale factor by which the numbers labeling the x-axis should be multiplied to get the correct graph units, that is, $\times 10^{\texttt{XPOW}}$ (see `NXDIG`)

The entire x-axis label is positioned in one of two ways:

- it is centred on the x-axis if the x-axis is at the bottom of the y-axis, or,
- it is centred on the part of the x-axis that is larger after being cut by the y-axis

The character string defined by `XLABEL` may contain text formatting commands which allow various changes within the string such as font changes, height changes, sub- and superscripts, special pre-defined characters, etc. See the section on **Text formatting**, page 89, for information on these formatting commands.

YLABEL

Default value: `YLABEL = ' '` (blank)

`YLABEL` controls the y-axis automatic text label. The maximum length of `YLABEL` is 255 characters. The complete y-axis text label is composed of two parts:

- the string defined by `YLABEL`, and
- the scale factor by which all the numbers labeling the y-axis should be multiplied to get the correct graph units, that is, $\times 10^{\texttt{YPOW}}$ (see `NYDIG`)

The entire y-axis label is positioned in one of two ways:

- it is centred on the y-axis if the y-axis is at the bottom of the x-axis, or,
- it is centred on the part of the y-axis that is larger after being cut by the x-axis

The character string defined by `YLABEL` may contain text formatting commands which allow various changes within the string such as font changes, height changes, sub- and super-scripts, special pre-defined characters, etc. See the **Text formatting** section, page 89, for information on these formatting commands.

13.5 Text formatting

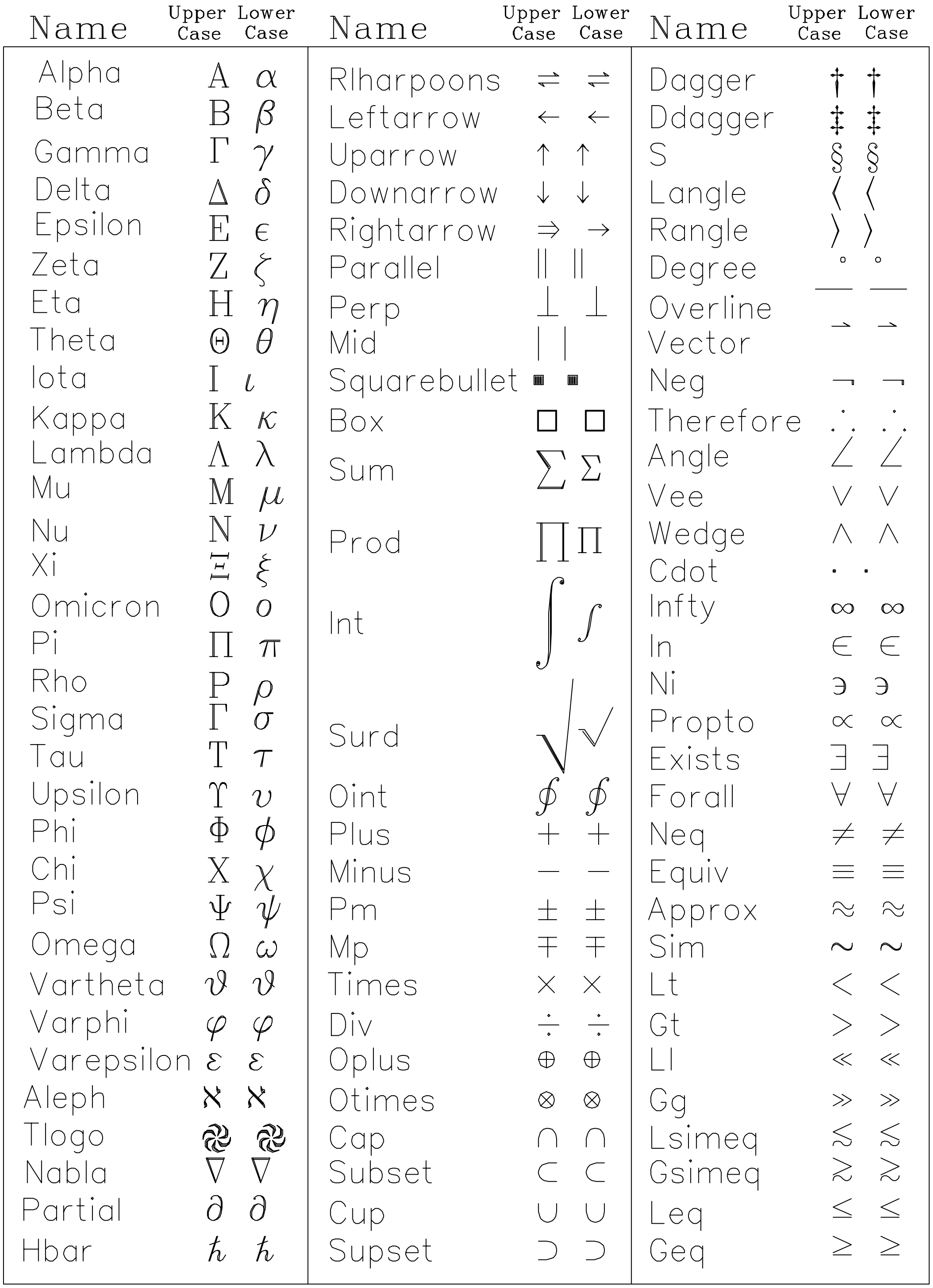
The character strings defined by `TEXT`, `XLABEL` and `YLABEL` may contain text formatting commands which allow various changes within the string such as font changes, height changes, sub- and super-scripts, special pre-defined characters, etc.

Text formatting commands must be bracketed by the command delimiters. The default command delimiters are < and >. The formatting commands are listed below.

Multiple commands can be entered within one set of delimiters by seperating the commands with commas. Some commands are expected to be followed by numbers, which may be real or integer. Any substring of the form <xxxxx> will be interpreted as a formatting command, thus, to *draw* a string of that form, use <LT>xxxxx<GT>. Also, a less than, <, not followed by a greater than, >, will be drawn directly.

command	*result*
`B`	select the fill pattern or turn filling off
`C`	select the colour
`F`	select the font
`H`	select the height
`_`	select sub-script mode (cancel super-script mode)
`^`	select super-script mode (cancel sub-script mode)
`EM`	toggle emphasis mode
`X`	toggle hexadecimal mode
`V`	insert a vertical space
`Z`	insert a horizontal space
`M`	insert a plotting symbol into the text string
`?`	display the menu of the formatting commands
`DEF`	reset font, bolding, colour, height, hexadecimal mode to the original defaults
`NOD`	do not reset the above defaults

SETLAB/GETLAB keywords



Name	Upper Case	Lower Case	Name	Upper Case	Lower Case	Name	Upper Case	Lower Case
Alpha	Α	α	Rlharpoons	⇌	⇌	Dagger	†	†
Beta	Β	β	Leftarrow	←	←	Ddagger	‡	‡
Gamma	Γ	γ	Uparrow	↑	↑	S	§	§
Delta	Δ	δ	Downarrow	↓	↓	Langle	⟨	⟨
Epsilon	Ε	ϵ	Rightarrow	⇒	→	Rangle	⟩	⟩
Zeta	Ζ	ζ	Parallel	‖	‖	Degree	°	°
Eta	Η	η	Perp	⊥	⊥	Overline	‾	‾
Theta	Θ	θ	Mid	∣	∣	Vector	⇀	⇀
Iota	Ι	ι	Squarebullet	▪	▪	Neg	¬	¬
Kappa	Κ	κ	Box	□	□	Therefore	∴	∴
Lambda	Λ	λ	Sum	∑	Σ	Angle	∠	∠
Mu	Μ	μ	Prod	∏	Π	Vee	∨	∨
Nu	Ν	ν	Int	∫	∫	Wedge	∧	∧
Xi	Ξ	ξ	Surd	√	√	Cdot	·	·
Omicron	Ο	ο	Oint	∮	∮	Infty	∞	∞
Pi	Π	π	Plus	+	+	In	∈	∈
Rho	Ρ	ρ	Minus	−	−	Ni	∋	∋
Sigma	Γ	σ	Pm	±	±	Propto	∝	∝
Tau	Τ	τ	Mp	∓	∓	Exists	∃	∃
Upsilon	ϒ	υ	Times	×	×	Forall	∀	∀
Phi	Φ	ϕ	Div	÷	÷	Neq	≠	≠
Chi	Χ	χ	Oplus	⊕	⊕	Equiv	≡	≡
Psi	Ψ	ψ	Otimes	⊗	⊗	Approx	≈	≈
Omega	Ω	ω	Cap	∩	∩	Sim	∼	∼
Vartheta	ϑ	ϑ	Subset	⊂	⊂	Lt	<	<
Varphi	φ	φ	Cup	∪	∪	Gt	>	>
Varepsilon	ε	ε	Supset	⊃	⊃	Ll	≪	≪
Aleph	ℵ	ℵ				Gg	≫	≫
Tlogo						Lsimeq	≲	≲
Nabla	∇	∇				Gsimeq	≳	≳
Partial	∂	∂				Leq	≤	≤
Hbar	ℏ	ℏ				Geq	≥	≥

Figure 13.19: Reserved character names in text

There is also a set of reserved character names, see Figure 13.19 on page 90. If the first character of the name is entered in uppercase, the uppercase form of the reserved character will be drawn, if the first character of the name is entered in lowercase, the lowercase reserved character is drawn. For example, entering `<Psi>` produces the character Ψ, while `<pSI>` produces the character ψ.

Note: The **GPLOT** reserved character name convention follows the LaTeX name convention.

Bolding

Bolding means that the characters will be filled, either with a hatch pattern or with dots. Bolding should only be used with the fillable fonts, `ROMAN.FUTURA`, `ROMAN.SERIF`, `ROMAN.SWISSL`, `ROMAN.SWISSM`, `ROMAN.SWISSB`, `ROMAN.FASHON`, `ROMAN.LOGO1`, and `TRIUMF.OUTLINE`. Those characters that have disconnected parts, for example, lower case 'i', 'j' and '%', will have an extrinsic connecting line in these fonts, *except* for the `TRIUMF.OUTLINE` font.

When `<Bn>` is encountered, subsequent characters will be filled. If `|n|` is between 1 and 10, the hatch pattern corresponding to the number `|n|` will be used. When `<Bn:m>` is encountered, two hatch patterns will be used to fill the characters. This can be used to create a cross-hatch effect. If `|n|` is greater than 10, a dot pattern will be used.

When `n` is negative, for example `<B-2>`, then just the fill pattern, using the absolute value of the fill number `n`, will be drawn and the outlines of the characters will not be drawn.

Within a string, characters will continue to be filled until `<B>` is encountered, which turns off bolding.

Hatch patterns

When drawing hatch patterns, the `HATCH_DRAW` routine is used. Refer to section **Hatch patterns**, page 51, for more information.

A hatch fill pattern is composed of an angle and one to ten spacings. The spacings are cycled through as each area is filled, that is, a line is drawn inside the area at the first spacing, then another parallel line is drawn at the second spacing, and so on for the number of spacings in that pattern. This process is repeated until the area is filled. The ten hatch patterns are listed in Table 10.24 on page 53. The spacings are expressed in the default world coordinate system units: $0 \le x \le 639$ and $0 \le y \le 479$, and the angles are in degrees. See Figure 10.12 on page 52 for examples of the default hatch patterns.

Dot patterns

$11 \leq |\mathrm{n}| \leq 99$ means to fill using a dot pattern. A dot pattern is of the form: uv, where the digit u is the increment number of dots to light up horizontally, $1 \leq u \leq 9$, and the digit v is the increment number of dots to light up vertically, $1 \leq v \leq 9$. For example, a dot pattern of 34 means to light up every third dot horizontally and every fourth dot vertically. If uv is negative, then the dots are erased instead of turned on.

Example

The following code produces Figure 13.20, where an X window monitor is chosen, along with PostScript hardcopy and no bitmap. The picture is drawn using portrait orientation.

```
CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
CALL CLEAR_PLOT
CALL HATCH_SCALE(1,0.02)
CALL HATCH_SCALE(7,0.01)
CALL HATCH_SCALE(8,0.01)
CALL SETNAM('%XLOC',50.)
CALL SETNAM('%YLOC',95.)
CALL SETNAM('CURSOR',-2.)
CALL SETLAB('TEXT','<FROMAN.SERIF,H3%,B1>Hatch bolding<NOD>')
CALL SETNAM('%YLOC',90.)
CALL SETLAB('TEXT','font = <FROMAN.LOGO1>ROMAN.LOGO1')
CALL SETNAM('%YLOC',85.)
CALL SETLAB('TEXT','Hatch pattern # 1')
CALL SETNAM('%YLOC',80.)
CALL SETLAB('TEXT','<B7:8>Hatch patterns # 7 # 8')
CALL NARGSI(1)
CALL GRAPHICS_HARDCOPY(0)
STOP
END
```

Colour

To set the colour code number, include `<Cn>` in the text, where $1 \leq \mathrm{n} \leq 11$. Within a string, this colour will remain until another `<Cn>` is encountered. Table 12.30 on page 73 details how the colour code number is mapped to actual colour.

Example

The following code fragment uses colour changes.

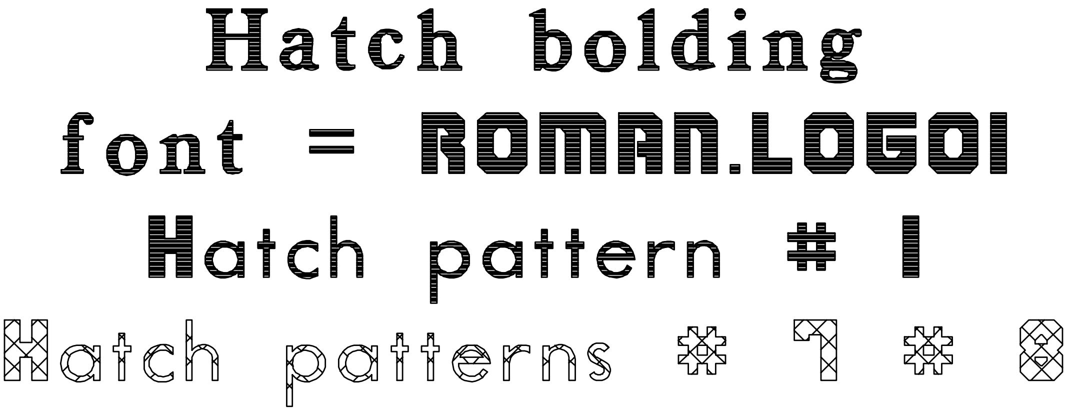


Figure 13.20: Text bolding example

```
...
CALL SETLAB('TEXT','<C1>Colour <C2>change <C3>example')
...
```

Font

To select a font for a string, include `<Ffontname>`. Table 10.22 on page 48 lists the currently available font names. Font tables are also available, the TRIUMF GRAPHIC FONTS manual, showing the characters and the hexadecimal code for each character.

Example

The following code produces Figure 13.21, where an X window monitor is chosen, along with PostScript hardcopy and no bitmap. The picture will be drawn in portrait orientation.

```
 CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
 CALL CLEAR_PLOT
 CALL SETNAM('%XLOC',50.)
 CALL SETNAM('%YLOC',95.)
 CALL SETNAM('CURSOR',-2.)
 CALL SETNAM('%TXTHIT',3.)
 CALL SETLAB('TEXT','<FROMAN.SWISSL>An example containing many fonts')
 CALL SETNAM('%YLOC',90.)
 CALL SETLAB('TEXT'
# ,'<FGOTHIC.ENGLISH>Gothic example <FSCRIPT.2>Script example')
 CALL SETNAM('%YLOC',85.)
 CALL SETLAB('TEXT','<FKANJI4>Kanji example')
 CALL SETNAM('%YLOC',80.)
 CALL SETLAB('TEXT','<FCYRILLIC.2>Cyrillic example')
 CALL SETNAM('%YLOC',70.)
 CALL SETLAB('TEXT','<FMATH>0123456789 SJKMOPQR')
 CALL NARGSI(1)
 CALL GRAPHICS_HARDCOPY(0)
 STOP
 END
```

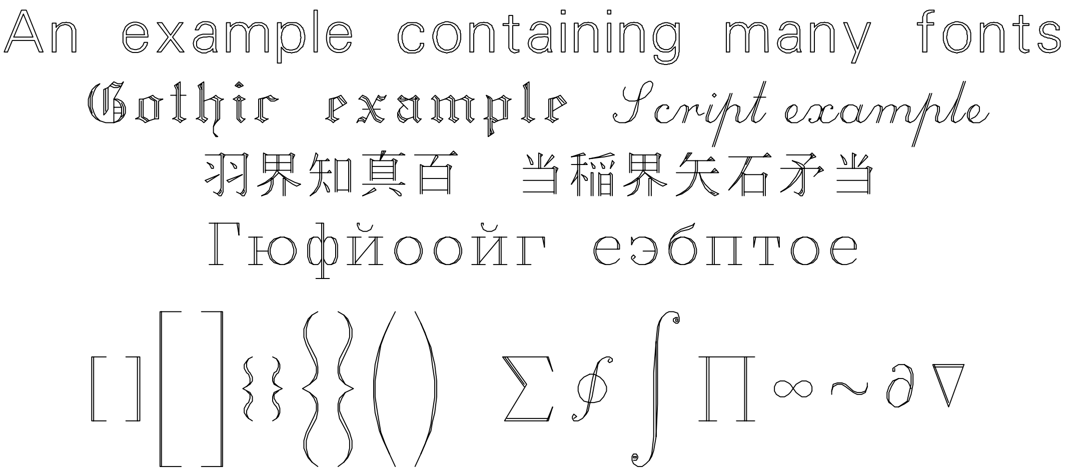

An example containing many fonts
Gothic example Script example
羽界知真百 当稲界矢石矛当
Гюфйоойг еэбптое
[ ] [ ] { } { } ( ) ∑ ∮ ∫ ∏ ∞ ∼ ∂ ∇

Figure 13.21: Text font example

Height

To set the character height within a string, include `<Hnn.n>` or `<Hnn.n%>`. When entered as a percent, it is a percentage of the height of the **GPLOT** window, that is, the horizontal space will be $\mathtt{nn.n} \times (\mathtt{YUWIND} - \mathtt{YLWIND}) \div 100$. When no percent sign is present, the units of `nn.n` are in the world coordinate system.

Example

The following code produces Figure 13.22, where an X window monitor is chosen, along with PostScript hardcopy and no bitmap. The picture will be drawn in portrait orientation.

```
CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
CALL CLEAR_PLOT
CALL SETNAM('%XLOC',50.)
```

```
      CALL SETNAM('%YLOC',95.)
      CALL SETNAM('CURSOR',-2.)
      CALL SETNAM('%TXTHIT',3.)
      CALL SETLAB('FONT','ROMAN.SWISSL')
      CALL SETLAB('TEXT','<H.5>C<H.6>h<H.7>a<H.9>r<H1.1>a<H1.4>c'//
     # '<H1.4>t<H1.1>e<H.9>r <H.7>h<H.6>e<H.5>ight can be')
      CALL SETNAM('%YLOC',85.)
      CALL SETLAB('TEXT','<H1>changed <H2>at <H1>any <H.5>time')
      CALL SETNAM('%YLOC',75.)
      CALL SETLAB('TEXT','but <H1>don''t<H.5> over do it')
      CALL NARGSI(1)
      CALL GRAPHICS_HARDCOPY(0)
      STOP
      END
```

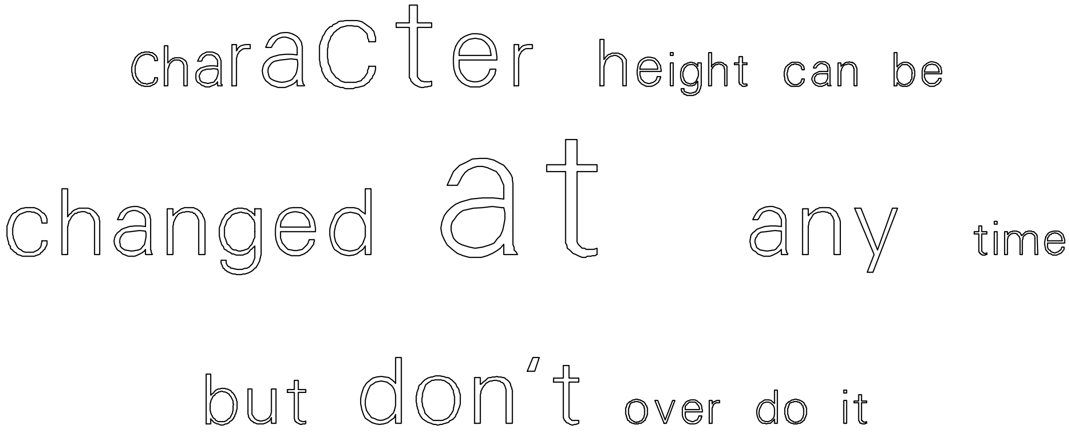


Figure 13.22: Text height example

Hexadecimal mode

To turn on hexadecimal mode, include `<X>` in the text. The first time `<X>` is encountered, hexadecimal mode is turned on. Subsequent text will be assumed to be pairs of hexadecimal digits that represent non-keyboard characters. The hexadecimal codes for characters depend on which font is being used. Refer to the font tables, the TRIUMF GRAPHIC FONTS manual, for these codes. A second `<X>` turns off hexadecimal mode.

The reserved character names, see Figure 13.19 on page 90, should eliminate the need for most usages of hexadecimal mode.

Example

SETLAB/GETLAB keywords

The following code produces Figure 13.23, where an X window monitor is chosen, along with PostScript hardcopy and no bitmap. The picture will be drawn in portrait orientation.

```
CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
CALL CLEAR_PLOT
CALL SETNAM('%XLOC',50.)
CALL SETNAM('%YLOC',95.)
CALL SETNAM('CURSOR',-2.)
CALL SETNAM('%TXTHIT',3.)
CALL SETLAB('FONT','TSAN')
CALL SETLAB('TEXT','Hexadecimal mode example')
CALL SETNAM('%YLOC',90.)
CALL SETLAB('TEXT','in the font <Rightarrow> TSAN')
CALL SETNAM('%YLOC',85.)
CALL SETLAB('TEXT','Greek letters: <X>CACBCCCDCECFDADBDCDDDEDF')
CALL SETNAM('%YLOC',80.)
CALL SETLAB('TEXT','and other <X>4AAFB96954555657<X> symbols')
CALL NARGSI(1)
CALL GRAPHICS_HARDCOPY(0)
STOP
END
```

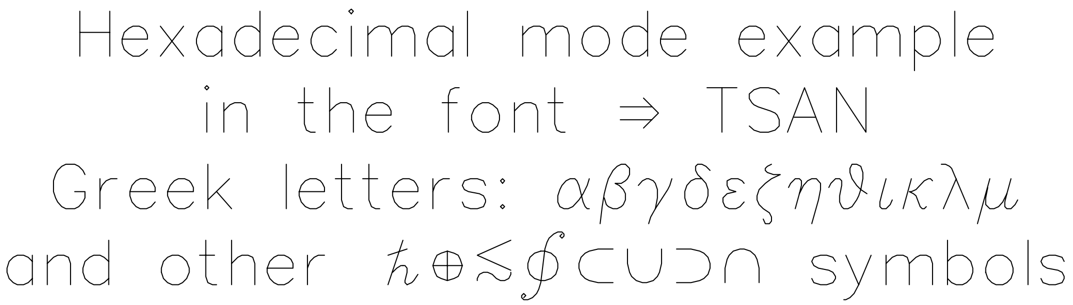


Figure 13.23: Hexadecimal mode example

Vertical spacing

To insert a vertical space, include `<Vnn.n>` or `<Vnn.n%>` in the text. When entered as a percent, it is a percentage of the height of the **GPLOT** window, that is, the vertical space will be $\mathtt{nn.n} \times (\mathtt{YUWIND} - \mathtt{YLWIND}) \div 100$. When no percent sign is present, the units of `nn.n` are in the world coordinate system. The space, which may be positive or negative, is measured from the current location.

Example

The following code produces Figure 13.24, where an X window monitor is chosen, along with PostScript hardcopy and no bitmap. The picture will be drawn in portrait orientation.

```
CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
CALL CLEAR_PLOT
CALL SETNAM('%XLOC',50.)
CALL SETNAM('%YLOC',85.)
CALL SETNAM('CURSOR',-2.)
CALL SETNAM('%TXTHIT',3.)
CALL SETLAB('FONT','TSAN')
CALL SETLAB('TEXT','Vertical spacing <V5%>can be <V-10%>changed')
CALL SETNAM('%YLOC',75.)
CALL SETLAB('TEXT','but the <V5%>spacing <V-10%>is <V5%>relative')
CALL SETNAM('%YLOC',65.)
CALL SETLAB('TEXT','to the <V-10%>current <V10%>location')
CALL NARGSI(1)
CALL GRAPHICS_HARDCOPY(0)
STOP
END
```

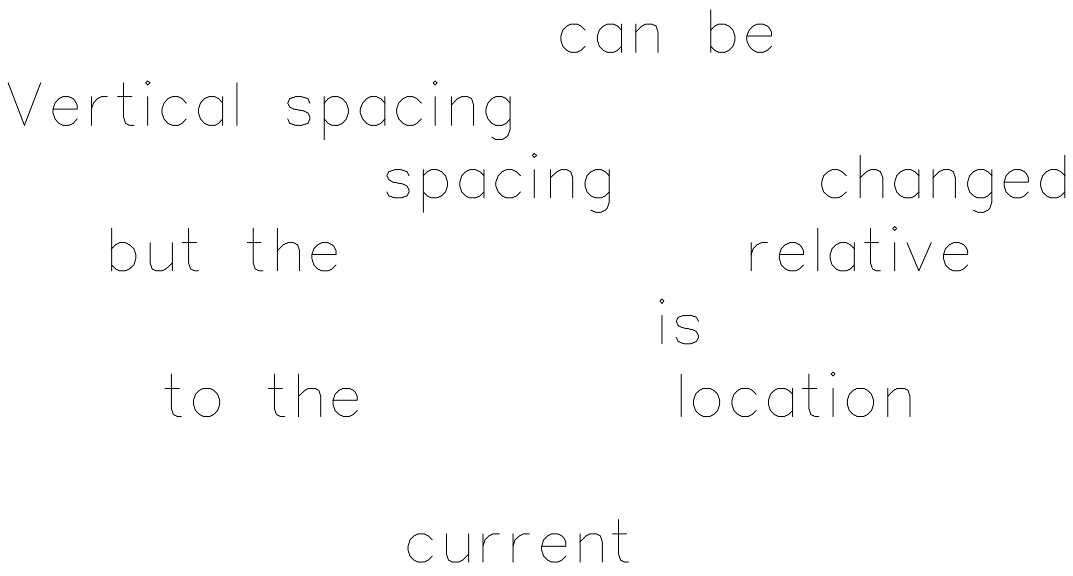


Figure 13.24: Text vertical spacing example

Horizontal spacing

To insert a horizontal space include <Znn.n> or <Znn.n%> in the text. When entered as a percent, it is a percentage of the width of the **GPLOT** window, that is, the horizontal space will be $\texttt{nn.n} \times (\texttt{XUWIND} - \texttt{XLWIND}) \div 100$. When no percent sign is present, the units of `nn.n` are in the world coordinate system. The space, which may be positive or negative, is measured from the current location.

Example

The following code produces Figure 13.25, where an X window monitor is chosen, along with PostScript hardcopy and no bitmap. The picture will be drawn in portrait orientation.

```
      CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
      CALL CLEAR_PLOT
      CALL SETNAM('%XLOC',5.)
      CALL SETNAM('%YLOC',95.)
      CALL SETNAM('CURSOR',-1.)
      CALL SETNAM('%TXTHIT',3.5)
      CALL SETLAB('FONT','TSAN')
      CALL SETLAB('TEXT','<H2%>Example of horizontal spacing')
      CALL SETNAM('%YLOC',88.)
      CALL SETLAB('TEXT','Insert a 2cm sp<Z2>ace')
      CALL SETNAM('%YLOC',75.)
      CALL SETLAB('TEXT','or move back-<Z-5,V4%>wards and up')
      CALL SETNAM('%YLOC',65.)
      CALL SETLAB('TEXT','or forwards <Z5%,V-4%>and down')
      CALL NARGSI(1)
      CALL GRAPHICS_HARDCOPY(0)
      STOP
      END
```

Slanted mode

To turn on slanted mode, include `<EM>` in the text. The first time `<EM>` is encountered, subsequent text will be emphasized, that is, slanted. The next time it is encountered, emphasis mode will be turned off. Emphasis mode may be used for any character in any font.

Note: This feature is *not* possible with **EDGR**, the graphics editor. So, if an **EDGR** file is opened and emphasised text is drawn, the emphasis will be lost when the graphics is replayed inside **EDGR**. The text will still be present, but it will not be emphasised.

Example

The following code produces Figure 13.26, where an X window monitor is chosen, along with

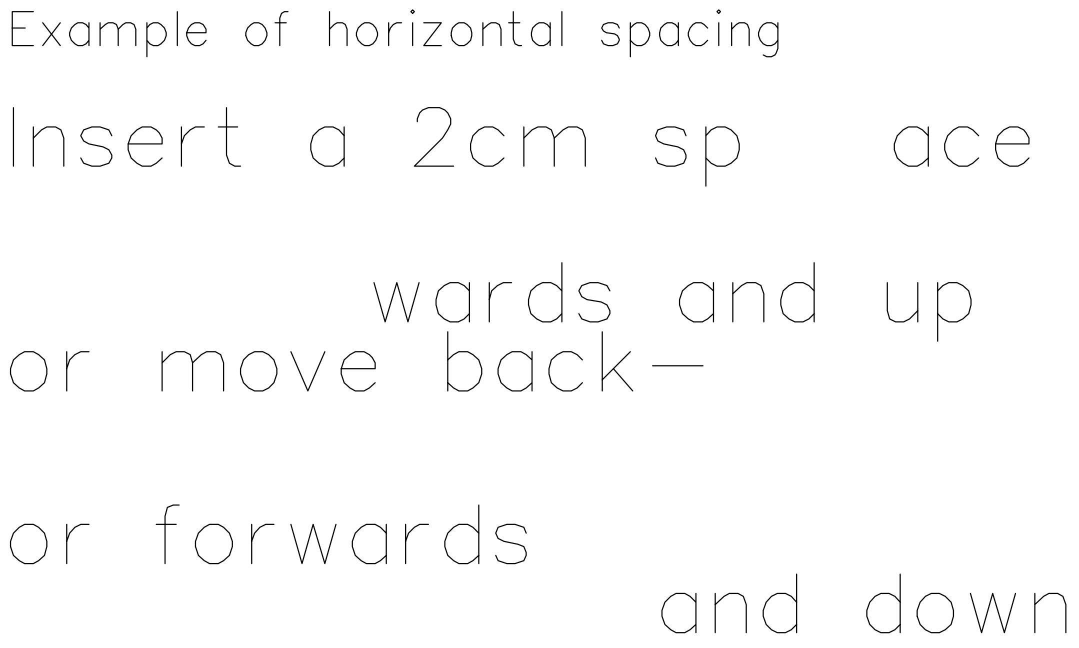


Figure 13.25: Text horizontal spacing example

PostScript hardcopy and no bitmap. The picture will be drawn in portrait orientation.

```
 CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
 CALL CLEAR_PLOT
 CALL SETNAM('%XLOC',5.)
 CALL SETNAM('%YLOC',95.)
 CALL SETNAM('CURSOR',-1.)
 CALL SETNAM('%TXTHIT',2.)
 CALL SETLAB('TEXT','<FTSAN,EM>Slanted<EM> in the TSAN font')
 CALL SETNAM('%YLOC',90.)
 CALL SETLAB('TEXT'
# ,'<FSTANDARD,EM>Slanted<EM> in the STANDARD font')
 CALL SETNAM('%YLOC',85.)
 CALL SETLAB('TEXT'
# ,'<FROMAN.SWISSL,EM>Slanted<EM> in the ROMAN.SWISSL font')
 CALL SETNAM('%YLOC',80.)
 CALL SETLAB('TEXT'
# ,'<FSCRIPT.2,EM>Slanted<EM> in the SCRIPT.2 font')
 CALL SETNAM('%YLOC',75.)
 CALL SETLAB('TEXT'
# ,'<FROMAN.SERIF,EM>Slanted<EM> in the ROMAN.SERIF font')
 CALL NARGSI(1)
 CALL GRAPHICS_HARDCOPY(0)
```

```
STOP
END
```

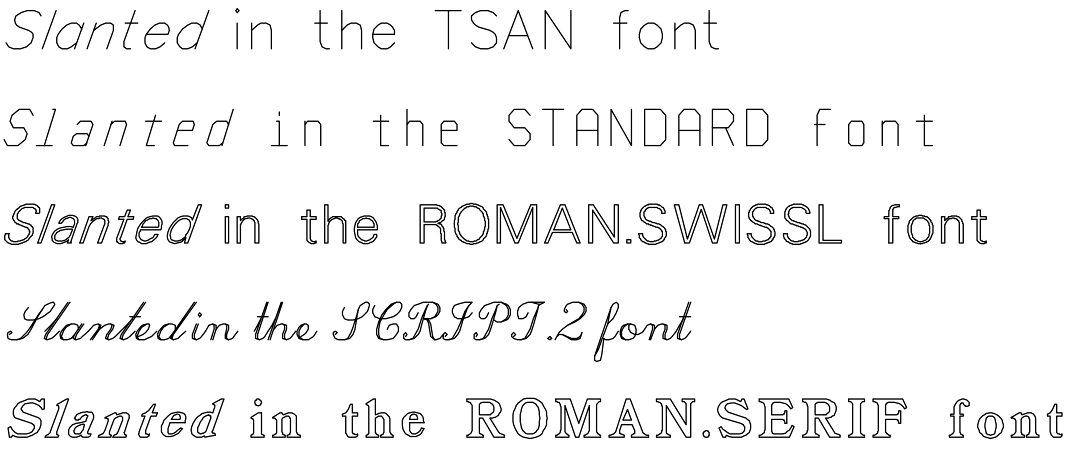

Slanted in the TSAN font
Slanted in the STANDARD font
Slanted in the ROMAN.SWISSL font
Slanted in the SCRIPT.2 font
Slanted in the ROMAN.SERIF font

Figure 13.26: Text slanted mode example

Sub-script mode

To enter sub-script mode, include <_> in the text. Subsequent text will have 60% of the current height and will be vertically spaced down a distance equal to 60% of the current height. Within a string, sub-script mode remains on until <^> is encountered.

Example

The following code produces Figure 13.27, where an X window monitor is chosen, along with PostScript hardcopy and no bitmap. The picture will be drawn in portrait orientation.

```
CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
CALL CLEAR_PLOT
CALL SETNAM('%XLOC',5.)
CALL SETNAM('%YLOC',95.)
CALL SETNAM('CURSOR',-1.)
CALL SETNAM('%TXTHIT',2.)
CALL SETLAB('FONT','TSAN')
CALL SETLAB('TEXT','Multiple levels of sub-scripts are allowed')
CALL SETNAM('%YLOC',90.)
CALL SETLAB('TEXT','Rember to go "up"')
CALL SETNAM('%YLOC',85.)
CALL SETLAB('TEXT','  as many times as you go "down"')
```

```
      CALL SETNAM('%YLOC',75.)
      CALL SETLAB('TEXT'
     # ,'<H6%,Upsilon,_,Theta,_,Psi,_>5<^,^,^,H2%>and back to normal')
      CALL NARGSI(1)
      CALL GRAPHICS_HARDCOPY(0)
      STOP
      END
```

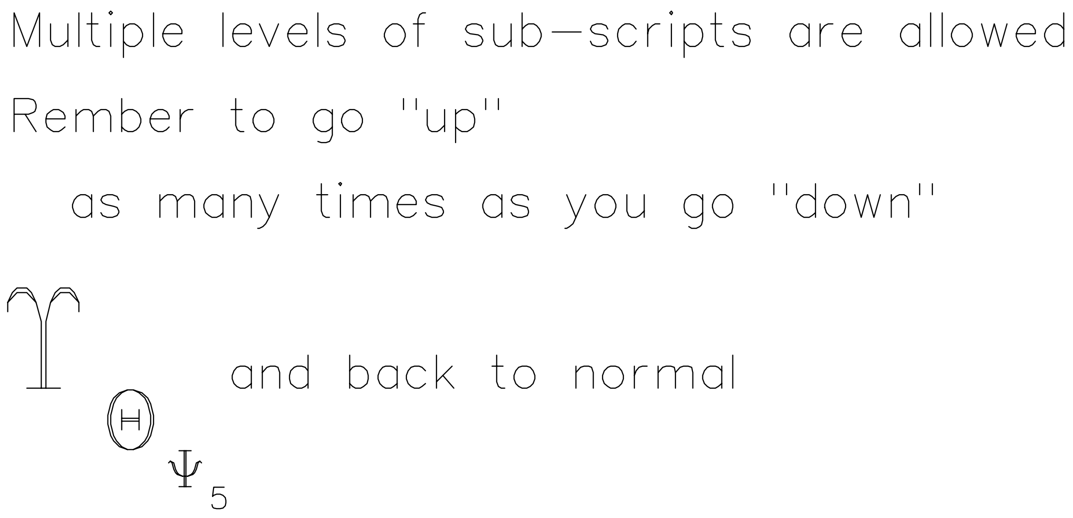

Multiple levels of sub-scripts are allowed
Rember to go "up"
as many times as you go "down"
Υ
Θ
Ψ
5
and back to normal

Figure 13.27: Text sub-script mode example

Super-script mode

To enter super-script mode, include <^> in the text. Subsequent text will have 60% of the current height and will be vertically spaced up a distance equal to 60% of the current height. Super-script mode remains on until <_> is encountered.

Example

The following code produces Figure 13.28, where an X window monitor is chosen, along with PostScript hardcopy and no bitmap. The picture will be drawn in portrait orientation.

```
      CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
      CALL CLEAR_PLOT
      CALL SETNAM('%XLOC',5.)
      CALL SETNAM('%YLOC',95.)
      CALL SETNAM('CURSOR',-1.)
      CALL SETNAM('%TXTHIT',2.)
      CALL SETLAB('FONT','TSAN')
      CALL SETLAB('TEXT','Multiple levels of super-scripts are allowed')
      CALL SETNAM('%YLOC',90.)
      CALL SETLAB('TEXT','Rember to go "down"')
      CALL SETNAM('%YLOC',85.)
      CALL SETLAB('TEXT','  as many times as you go "up"')
      CALL SETNAM('%YLOC',75.)
      CALL SETLAB('TEXT'
     # ,'<H6%,Upsilon,^,Theta,^,Psi,^>5<_,_,_,H2%>and back to normal')
      CALL NARGSI(1)
      CALL GRAPHICS_HARDCOPY(0)
      STOP
      END
```

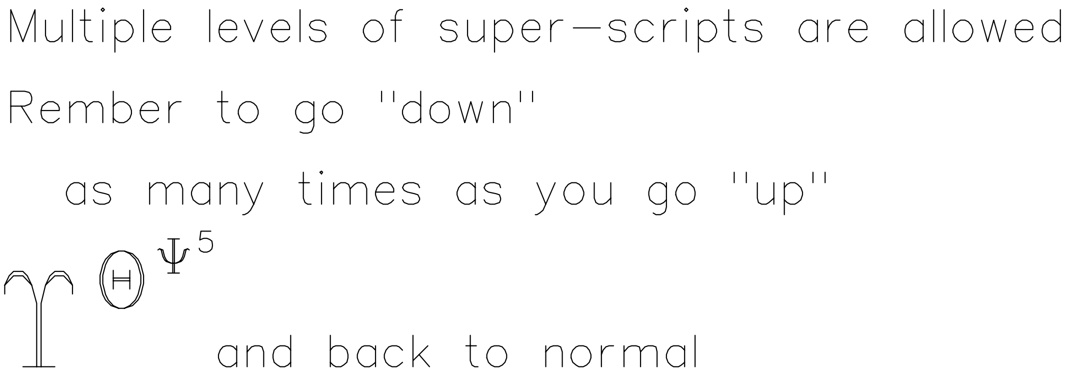

Multiple levels of super-scripts are allowed
Rember to go "down"
as many times as you go "up"
$\Upsilon^{\Theta^{\Psi^5}}$
and back to normal

Figure 13.28: Text super-script mode example

Inserting a plotting symbol into text

To insert a plotting symbol, include `<Mn>` in the text. If $0 < \texttt{n} < 32$, the special plotting symbol corresponding to `n` will be inserted in the text. See Figure 13.18 for examples of the special plotting symbols. When $\texttt{n} = 0$, the current plotting symbol number is used,[11] while if $\texttt{n} = 32$, no symbol will be used. If $\texttt{n} > 32$, the ASCII character corresponding to the decimal code `n` will be inserted into the text. The size of the plotting symbol included in the text string is `CHARSZ` and not `TXTHIT`.

Example

---

[11] The current plotting symbol is found from `CHAR(1:1)` as follows:

```
      CHARACTER*2 GETLAB, CHARL
      CHARL = GETLAB('CHAR')
      N = ICHAR(CHARL(1:1))
```

The following code produces Figure 13.29, where an X window monitor is chosen, along with PostScript hardcopy and no bitmap. The picture will be drawn in portrait orientation.

```
CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
CALL CLEAR_PLOT
CALL SETNAM('%XLOC',5.)
CALL SETNAM('%YLOC',95.)
CALL SETNAM('CURSOR',-1.)
CALL SETNAM('%TXTHIT',2.)
CALL SETLAB('FONT','TSAN')
CALL SETNAM('%CHARSZ',3.)
CALL SETLAB('TEXT','Insert a "star" symbol <M14> into a string')
CALL SETNAM('%YLOC',90.)
CALL SETLAB('TEXT','Insert a "box" symbol <M1> into a string')
CALL SETNAM('%YLOC',85.)
CALL SETLAB('TEXT','Insert a "filled circle" symbol <M13> into a string')
CALL NARGSI(1)
CALL GRAPHICS_HARDCOPY(0)
STOP
END
```

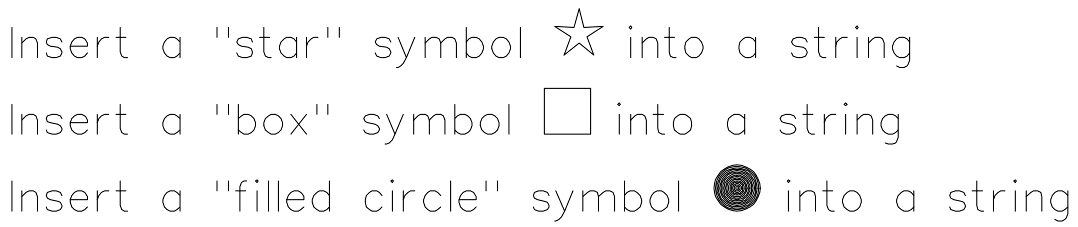

Insert a "star" symbol ☆ into a string
Insert a "box" symbol ☐ into a string
Insert a "filled circle" symbol ● into a string

Figure 13.29: Inserting a plotting symbol into text

Reset the defaults

When <DEF> is included in a text string, after processing that string, bolding will be turned off, the colour will be reset to the last colour chosen by some means other than <Cn>, the font will be reset to the last font chosen by some means other than <Ffont>, emphasis mode will be turned off, and hexadecimal mode will be turned off. This is the default action, so it is not necessary to include <DEF> in a text string. It has been included mainly for completeness.

Do not reset the defaults

When <NOD> is included in a text string, the bolding, colour, font, emphasis and hexadecimal mode will be left in their current state after processing that string.

14 SETNAM/GETNAM KEYWORDS

The keywords refered to in this chapter are the keywords that can be passed to the routines `GETNAM` and `SETNAM`. These entities control the appearance of a graph.

The values associated with *every* name are `REAL*4`, and all angles are in degrees.

14.1 Summary

The following is a list of keywords, with very terse descriptions and their `GPLOTI` default values.

General keywords

name	*description*	*default*
`PTYPE`	controls whether pixels are turned on, off, or complemented	0
`LINTYP`	line type for joining data points or a filling pattern	1
`LINTHK`	line thickness	1
`COLOUR`	colour	1
`NUMBLD`	fill number for the axis numbers	0
`CLIP`	controls whether data curves are clipped at the box edge	1
`HISTYP`	histogram type	0
`XLWIND`	left edge of the **GPLOT** window	0%
`XUWIND`	right edge of the **GPLOT** window	100%
`YLWIND`	bottom edge of the **GPLOT** window	0%
`YUWIND`	top edge of the **GPLOT** window	100%

Text keywords

name	*description*	*default*
`CURSOR`	text justification code	1
`TXTANG`	text angle	0
`TXTHIT`	text height	3%
`XLOC`	horizontal reference location for text positioning	50%
`YLOC`	vertical reference location for text positioning	50%

Axis box keywords

SETNAM/GETNAM keywords

name	*description*	*default*
`BOX`	controls whether or not to draw an axis box around the graph	1
`XLAXIS`	location of the lower end of the x-axis	15%
`XUAXIS`	location of the upper end of the x-axis	95%
`XAXISA`	angle of the x-axis	0
`YLAXIS`	location of the bottom of the y-axis	15%
`YUAXIS`	location of the top of the y-axis	90%
`YAXISA`	angle of the y-axis	90
`BOTNUM`	controls the height of the numbers on the bottom of the box	0
`BOTTIC`	controls the length of the tic marks on the bottom of the box	1
`RITNUM`	controls the height of the numbers on the right side of the box	0
`RITTIC`	controls the length of the tic marks on the right side of the box	-1
`TOPNUM`	controls the height of the numbers on the top of the box	0
`TOPTIC`	controls the length of the tic marks on the top of the box	-1
`LEFNUM`	controls the height of the numbers on the left side of the box	0
`LEFTIC`	length of the tic marks on the left side of the box	1

Plotting symbol keywords

name	*description*	*default*
`MASK`	plotting symbol mask	0
`PMODE`	controls whether data points are to be joined or unjoined	1
`CHARA`	plotting symbol angle	horizontal
`CHARSZ`	plotting symbol size	1%
`ERRBAR`	type of error bars	0

SETNAM/GETNAM keywords

x-axis keywords

name	*description*	*default*
`XAXIS`	controls whether or not to draw the x-axis	1
`XLABSZ`	height of the x-axis text label	3%
`XLOG`	base of the x-axis numbers	0
`NXGRID`	number of grid lines to draw parallel to the y-axis	0
`XCROSS`	controls where the y-axis will cross the x-axis	0
`XZERO`	controls whether zero is forced to appear on the x-axis	0
`XTICTP`	type of tic marks to place on the x-axis	1
`XTICA`	angle of the x-axis tic marks	$\parallel$ y-axis
`NLXINC`	number of long x-axis tic marks	2
`XTICL`	length of the long tic marks on the x-axis	2%
`NSXINC`	number of short x-axis tic marks	1
`XTICS`	length of the short tic marks on the x-axis	1%
`XMAX`	maximum value for the x-axis	10
`XVMAX`	virtual maximum for the x-axis	10
`XMIN`	minimum value for the x-axis	0
`XVMIN`	virtual minimum for the x-axis	0
`XMOD`	base of the modulus for x-axis numbering	0
`XOFF`	offset added to the numbers labeling the x-axis	0
`XLEADZ`	controls whether x-axis leading zeros are displayed	0
`XPAUTO`	controls the automatic x-axis scale factor	1
`XPOW`	x-axis numbers scale factor	0
`NXDIG`	number of digits to display in the x-axis numbers	5
`NXDEC`	number of decimal places to display in the x-axis numbers	-1
`XNUMSZ`	height of the numbers labeling the x-axis	3%
`XNUMA`	angle of the numbers labeling the x-axis	horizontal
`XITICA`	angle at which to position the x-axis numbers	$\parallel$ y-axis
`XITICL`	distance from the x-axis to the numbers labeling the x-axis	3%

y-axis keywords

name	*description*	*default*
YAXIS	controls whether or not to draw the y-axis	1
YLABSZ	height of the y-axis text label	3%
YLOG	controls whether the y-axis is to be linear or logarithmic	0
NYGRID	number of grid lines to draw parallel to the x-axis	0
YCROSS	controls where the x-axis will cross the y-axis	0
YZERO	controls whether zero is forced to appear on the y-axis	0
YTICTP	type of tic marks to place on the y-axis	1
YTICA	angle of the y-axis tic marks	$\parallel$ x-axis
NLYINC	number of long y-axis tic marks	2
YTICL	length of the long tic marks on the y-axis	2%
NSYINC	number of short y-axis tic marks	1
YTICS	length of the short tic marks on the y-axis	1%
YMAX	maximum value for the y-axis	10
YVMAX	virtual maximum for the y-axis	10
YMIN	minimum value for the y-axis	0
YVMIN	virtual minimum value for the y-axis	0
YMOD	base of the modulus for the y-axis numbering	0
YOFF	offset added to the numbers labeling the y-axis	0
YLEADZ	controls whether y-axis leading zeros are displayed	0
YPAUTO	controls the automatic y-axis scale factor	1
YPOW	y-axis numbers scale factor	0
NYDIG	number of digits to display in the y-axis numbers	5
NYDEC	number of decimal places to display in the y-axis numbers	-1
YNUMSZ	height of the numbers labeling the y-axis	3%
YNUMA	angle of the numbers labeling the y-axis	horizontal
YITICA	angle at which to position the y-axis numbers	$\parallel$ y-axis
YITICL	distance from the y-axis to the numbers labeling the y-axis	3%

14.2 General keywords

PTYPE

Default value: PTYPE $= 0$

PTYPE controls whether pixels on the terminal screen are turned on, off, or toggled. This can be used to selectively erase graphics, by drawing with PTYPE $= 0$ and redrawing with PTYPE

$= 1$; or by drawing and redrawing with `PTYPE` $= 2$. See also the section on **Selective erasing or complementing**, page 56.

This only works on the monitor screen and bitmap output.

`PTYPE` $= 0$	pixels turned on (draw)
`PTYPE` $= 1$	pixels turned off (erase)
`PTYPE` $= 2$	pixels complemented

Complemented pixels means that a pixel is turned off if it is on, or turned off if it is on.

LINTYP

Default value: `LINTYP` $= 1$

`LINTYP` either controls the type of line to use when drawing a data curve or it indicates the pattern to use for filling the area under a data curve.

14.2.0.1 Line types

If $1 \leq$ `LINTYP` ≤ 10, then line type number `LINTYP` is chosen. The routines used to draw the dashed lines are explained in section **Dashed lines**, page 44. See Figure 10.11 on page 44 for examples of the default line types. The length of the dashes in the dashed line types, the default types 3 to 10, are constant and do not depend on the seperation between data points. The default line types are suitable for a 640×480 coordinate system. The line types can be scaled to fit a user defined world coordinate system by calling the `DLINESCALE` routine. The definition of any one line type can be changed by calling the `DLINESET` routine.

14.2.0.2 Hatch patterns

If $101 \leq |$`LINTYP`$| \leq 110$, then the hatch pattern $|$`LINTYP`$| - 100$ is chosen. The polygonal region defined by the (x, y) coordinate pairs that are passed in a subsequent call to `GPLOT` will be filled with the chosen hatch pattern. The last point is automatically connected to the first point for closure.

The following example program produces Figure 14.30 on page 110.

```
REAL*4 X(198), Y(198)
CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
CALL CLEAR_PLOT
CALL HATCH_SCALE(5,0.05)
CALL HATCH_SCALE(7,0.05)
CALL HATCH_SCALE(8,0.05)
DO I = 1, 100
  X(I) = FLOAT(I-1)
  Y(I) = 3.0*SIN(X(I)*3.14159/90.)
END DO
DO I = 101, 198
  X(I) = FLOAT(199-I)
  Y(I) = SIN(X(I)*3.14159/60.)
END DO
CALL SETNAM('%XNUMSZ',5.)
CALL SETNAM('%YNUMSZ',5.)
CALL SETNAM('%YLWIND',50.)
CALL SETNAM('LINTYP',105.)
CALL NARGSI(4)
CALL GPLOT(X,Y,198,1)
X(1) = 1.
Y(1) = 0.
DO I = 2, 10
  X(I) = FLOAT(I)
  Y(I) = SIN(X(I)*3.14159/5.)
END DO
X(11) = 11.
Y(11) = 0.
CALL SETNAM('HISTYP',1.)
CALL SETNAM('%YUWIND',50.)
CALL SETNAM('%YLWIND',0.)
CALL SETNAM('LINTYP',107.)
CALL SETNAM('XAUTO',2.)
CALL SETNAM('YAUTO',2.)
CALL NARGSI(4)
CALL GPLOT(X,Y,11,1)
CALL SETNAM('LINTYP',108.)
CALL NARGSI(4)
CALL GPLOT(X,Y,11,2)
CALL NARGSI(1)
CALL GRAPHICS_HARDCOPY(0)
STOP
END
```

Refer to section **Hatch patterns**, page 51, for more detailed information. A hatch fill pattern

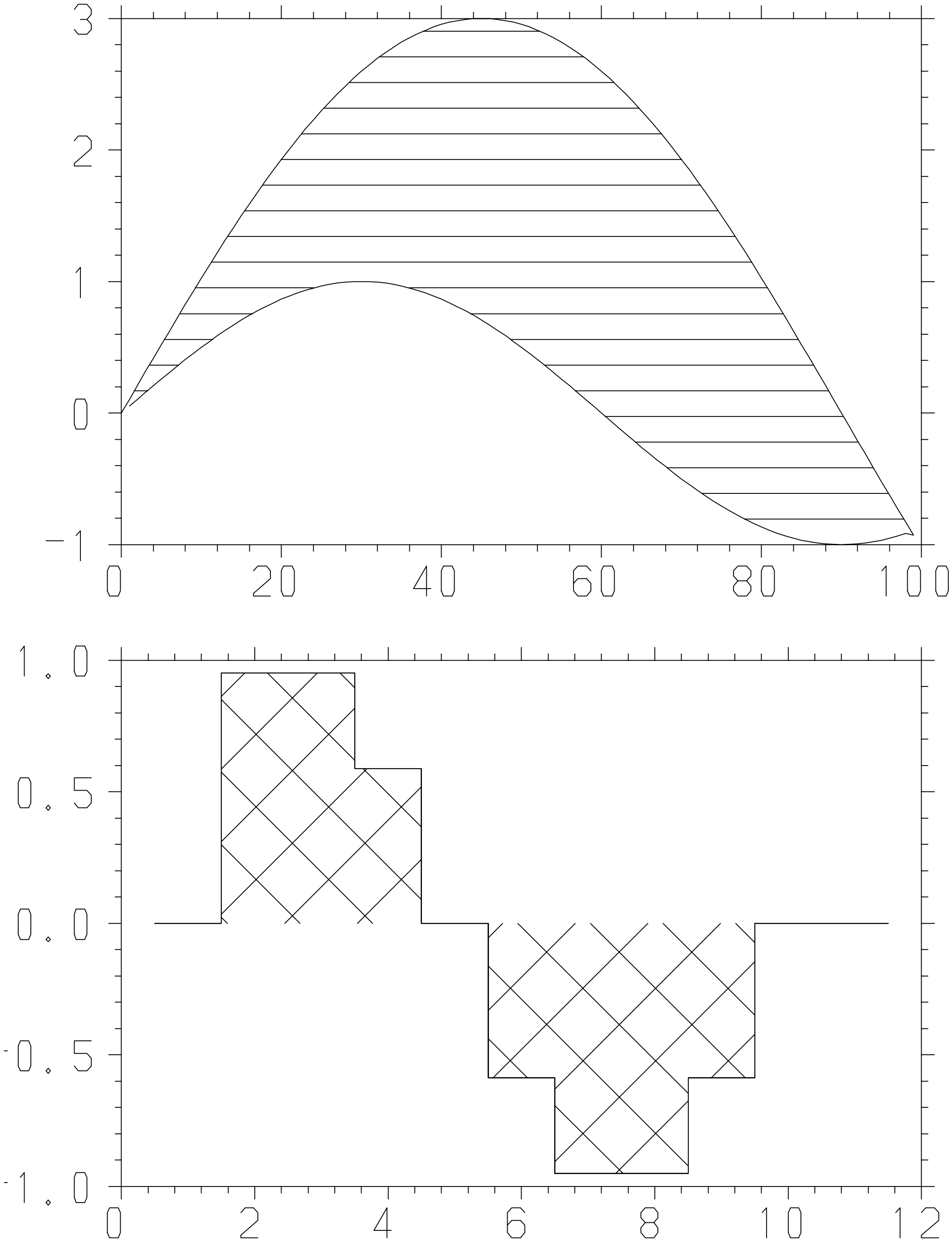


Figure 14.30: Hatch fill example

is composed of an angle and one to ten spacings. The spacings are cycled through as each area is filled, that is, a line is drawn inside the area at the first spacing, then another parallel line is drawn at the second spacing, and so on for the number of spacings in that pattern. This process is repeated until the area is filled. The ten hatch patterns are listed in Table 10.24 on page 53. The spacings are expressed in the default world coordinate system units: $0 \leq x \leq 639$ and $0 \leq y \leq 479$, and the angles are in degrees. See Figure 10.12 on page 52 for examples of the default hatch patterns. The default hatch patterns are suitable for a 640×480 coordinate system. The hatch patterns can be scaled to fit a user defined world coordinate system by calling the `HATCH_SCALE` routine. The definition of any one hatch pattern can be changed by calling the `HATCH_SET` routine. The `HATCH_GET` routine has the same parameters as the `HATCH_SET` routine, but the current spacings are returned.

14.2.0.3 Dot patterns

If $211 \leq |\texttt{LINTYP}| \leq 299$, then the dot pattern $|\texttt{LINTYP}| - 200$ is chosen. The polygonal region defined by the (x, y) coordinate pairs that are passed in a subsequent call to `GPLOT` will be filled with the chosen dot pattern. The last point is automatically connected to the first point for closure. See Figure 14.31 on page 113 for a dot fill example.

The following example program produces Figure 14.31 on page 113.

```
REAL*4 X(198), Y(198)
CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
CALL CLEAR_PLOT
DO I = 1, 100
  X(I) = FLOAT(I-1)
  Y(I) = 3.0*SIN(X(I)*3.14159/90.)
END DO
DO I = 101, 198
  X(I) = FLOAT(199-I)
  Y(I) = SIN(X(I)*3.14159/60.)
END DO
CALL SETNAM('%XNUMSZ',5.)
CALL SETNAM('%YNUMSZ',5.)
CALL SETNAM('%YLWIND',50.)
CALL SETNAM('LINTYP',233.)
CALL NARGSI(4)
CALL GPLOT(X,Y,198,1)
X(1) = 1.
Y(1) = 0.
DO I = 2, 10
  X(I) = FLOAT(I)
  Y(I) = SIN(X(I)*3.14159/5.)
END DO
```

```
      X(11) = 11.
      Y(11) = 0.
      CALL SETNAM('HISTYP',1.)
      CALL SETNAM('%YUWIND',50.)
      CALL SETNAM('%YLWIND',0.)
      CALL SETNAM('LINTYP',255.)
      CALL SETNAM('XAUTO',2.)
      CALL SETNAM('YAUTO',2.)
      CALL NARGSI(4)
      CALL GPLOT(X,Y,11,1)
      CALL NARGSI(1)
      CALL GRAPHICS_HARDCOPY(0)
      STOP
      END
```

A dot pattern is of the form: uv, where the digit u is the increment number of dots to light up horizontally, $1 \leq u \leq 9$, and the digit v is the increment number of dots to light up vertically, $1 \leq v \leq 9$. For example, a dot pattern of 34 means to light up every third dot horizontally and every fourth dot vertically. If uv is negative, then the dots are erased instead of turned on.

LINTHK

Default value: `LINTHK` $= 1$

`LINTHK` controls the line thickness for bitmap hardcopy output and for PostScript hardcopy output. `LINTHK` has no affect on monitor screen output or on pen plotter hardcopy output.

14.2.0.1 Bitmap hardcopies

The following applies to bitmap hardcopies only. The line width in pixels is set by `LINTHK`. The “brush” used is a square of `LINTHK` $\times$ `LINTHK` pixels. The round brush option is enabled by making `LINTHK` negative and makes a difference only with widths of 4 pixels or more. Compared to the square brush, the round brush gives a more consistent line width for lines of different angles.

COLOUR

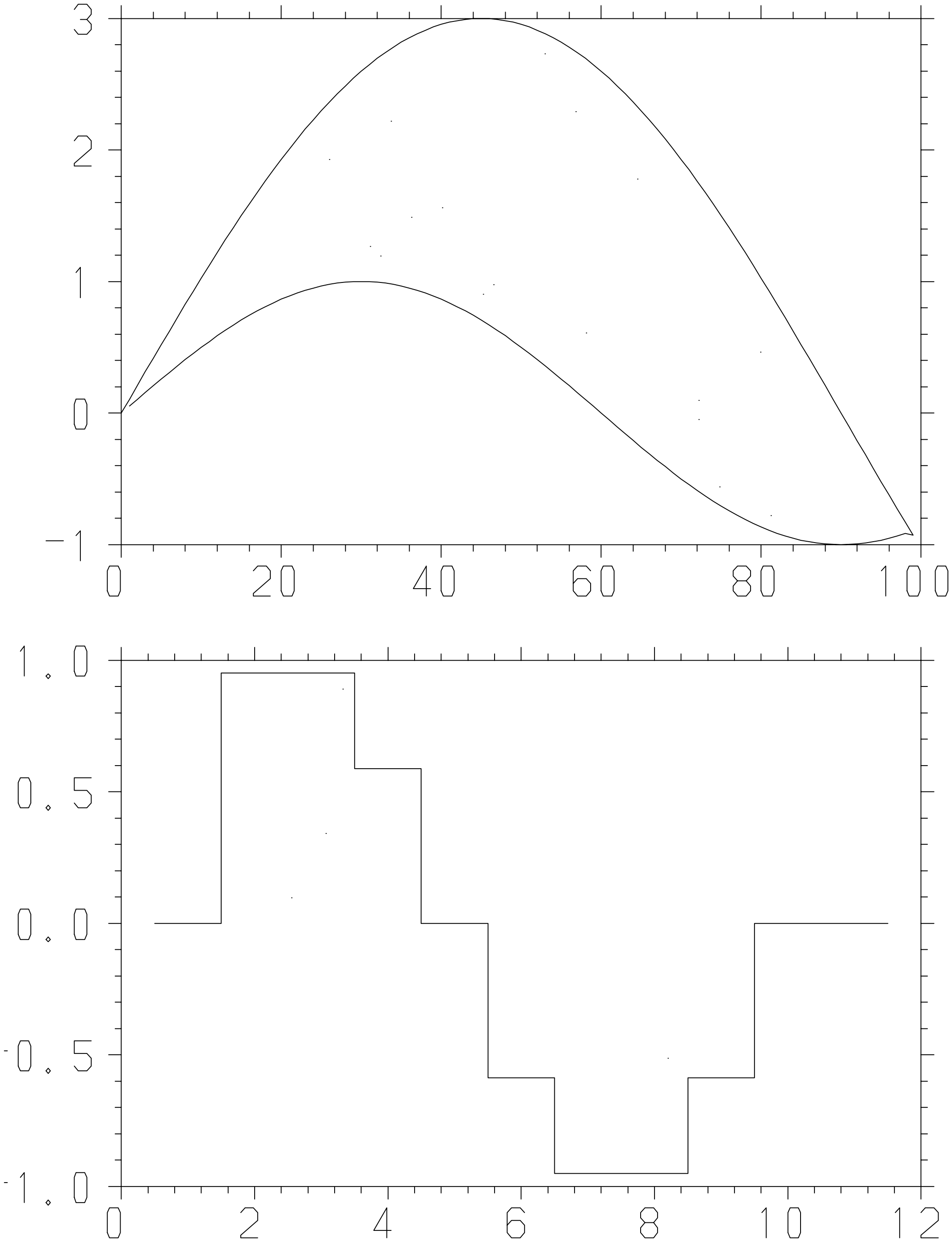


Figure 14.31: Dot fill example

Default value: `COLOUR` $= 1$

`COLOUR` sets a colour code which is used to control the monitor colour and the hardcopy colour. This colour will become the default colour, just as if the `PLOT_COLOR` routine had been called. Following is a list of the colour codes and their corresponding colours.

`COLOUR`	*colour*	*plotter pen*	
0	black	—	
1	white	black	(thick pen)
2	red	red	(thick pen)
3	green	green	(thick pen)
4	blue	blue	(thick pen)
5	yellow	black	(thin pen)
6	cyan	red	(thin pen)
7	magenta	green	(thin pen)
8	coral	blue	(thin pen)
9	red magenta	—	—
10	green cyan	—	—
11	blue cyan	—	—

NUMBLD

Default: `NUMBLD` $= 0$

`NUMBLD` is the fill number for the axis numbers.

`NUMBLD` $= 0$	no filling
$1 \leq$ `NUMBLD` ≤ 10	use hatch pattern
$11 \leq$ `NUMBLD` ≤ 99	use dot fill pattern

CLIP

Default value: `CLIP` $= 1$

`CLIP` controls whether or not data curves that are plotted are clipped at the boundaries of the axis box.

`CLIP` $= 0$ do not clip

`CLIP` $\neq 0$ clip at the boundaries of the axis box

The following example program produces Figure 14.32 on page 116.

```
REAL*4 X(100), Y(100)
CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
CALL CLEAR_PLOT
DO I = 1, 100
  X(I) = FLOAT(I-1)
  Y(I) = COS(X(I)*3.14159/50.)
END DO
CALL SETNAM('%YLWIND',50.)
CALL SETNAM('XAUTO',0.)
CALL SETNAM('YAUTO',0.)
CALL SETNAM('XMIN',10.)
CALL SETNAM('XMAX',90.)
CALL SETNAM('YMIN',-0.8)
CALL SETNAM('YMAX',0.8)
CALL NARGSI(4)
CALL GPLOT(X,Y,100,1)
CALL SETNAM('%XLOC',50.)
CALL SETNAM('%YLOC',80.)
CALL SETNAM('CURSOR',-2.)
CALL SETLAB('TEXT','CLIP = 1 (default)')
CALL SETNAM('%YUWIND',50.)
CALL SETNAM('%YLWIND',0.)
CALL SETNAM('CLIP',0.)
CALL NARGSI(4)
CALL GPLOT(X,Y,100,1)
CALL SETLAB('TEXT','CLIP = 0')
CALL NARGSI(1)
CALL GRAPHICS_HARDCOPY(0)
STOP
END
```

HISTYP

Default: `HISTYP` $= 0$

`HISTYP` controls whether a normal line graph or a histogram is drawn. Histograms can have tails or no tails and the profile can be along the x-axis or along the y-axis.

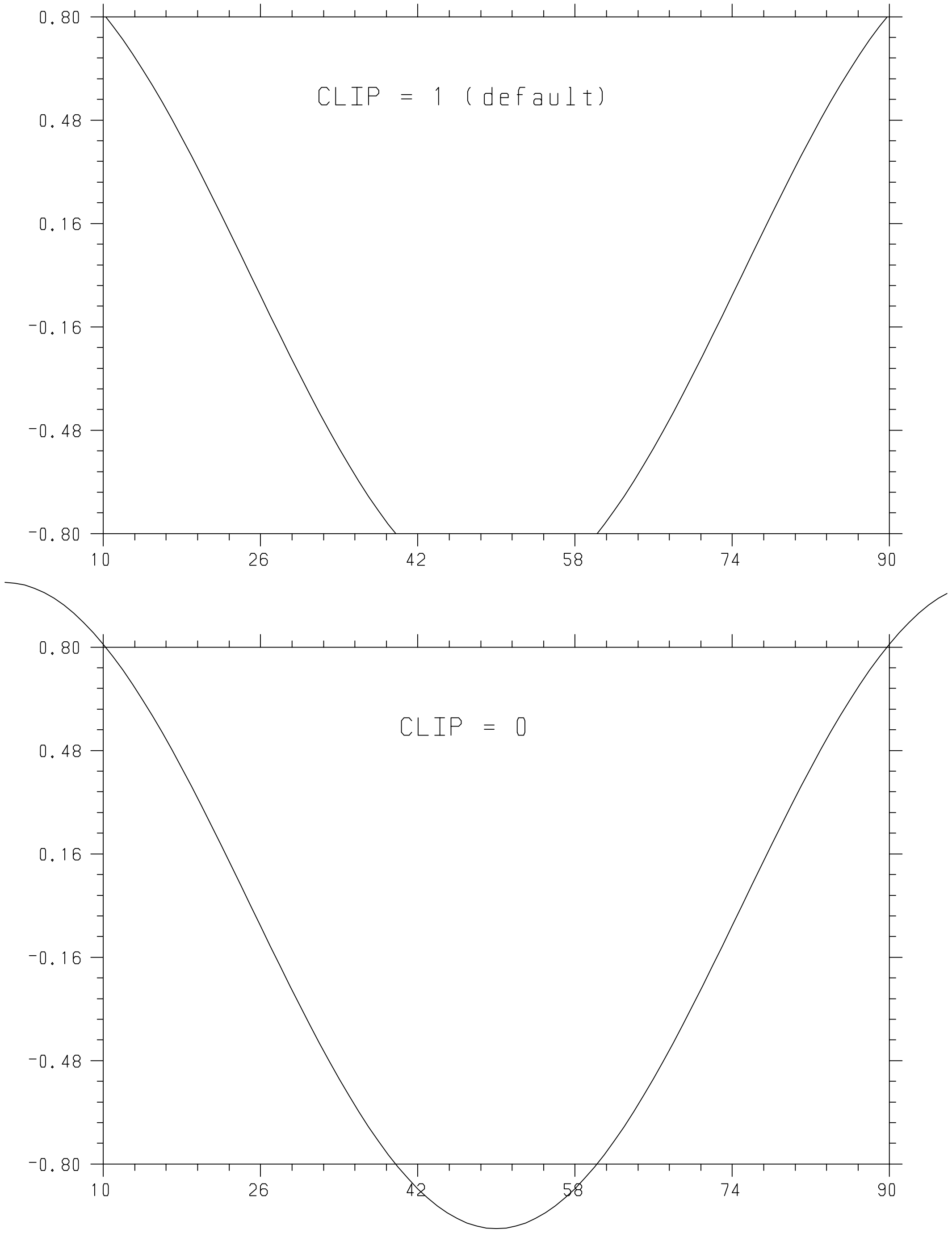


Figure 14.32: Clipping example

HISTYP $= 0$	normal line graph, not a histogram
HISTYP $= 1$	histogram with no tails and profile along the x-axis. Can control the width and colour of each individual bar
HISTYP $= 2$	histogram with tails to $y = 0$ and profile along the x-axis. Can control the filling pattern, width and colour of each individual bar
HISTYP $= 3$	histogram without tails and profile along the y-axis. Can control the height and colour of each individual bar
HISTYP $= 4$	histogram with tails to $x = 0$ and profile along the y-axis. Can control the filling pattern, height and colour of each individual bar

No plotting symbol will be drawn at the data points when HISTYP > 0. The fill pattern, width and colour of each histogram bar can be controlled by using the arrays $pchar$, $psize$ and $pcolr$, which are optional arguments for the GPLOT routine. The interpretation of these optional arguments is dependent on the value of MASK. Table 14.40 on page 134 describes how MASK is interpreted when HISTYP > 0. Table 12.29 on page 72 describes the relationship of HISTYP to the size of the histogram bars.

The following program produces Figure 14.33 on page 118.

```
REAL*4 X(20), Y(20)
CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
CALL CLEAR_PLOT
ISEED = 12345678
DO I = 1, 20
  X(I) = FLOAT(I)
  Y(I) = 4.*RAN(ISEED)-2.0
END DO
CALL SETNAM('%YLWIND',50.)
CALL SETNAM('%XUWIND',50.)
CALL SETNAM('HISTYP',1.)
CALL SETLAB('XLABEL','HISTYP = 1')
CALL NARGSI(4)
CALL GPLOT(X,Y,20,1)
CALL SETNAM('%XUWIND',100.)
CALL SETNAM('%XLWIND',50.)
CALL SETNAM('HISTYP',2.)
CALL SETLAB('XLABEL','HISTYP = 2')
CALL NARGSI(4)
CALL GPLOT(X,Y,20,1)
CALL SETNAM('YAUTO',2.)
CALL SETNAM('XAUTO',2.)
CALL SETNAM('%XUWIND',50.)
CALL SETNAM('%XLWIND',0.)
CALL SETNAM('%YUWIND',50.)
CALL SETNAM('%YLWIND',0.)
CALL SETNAM('HISTYP',3.)
CALL SETLAB('XLABEL','HISTYP = 3')
CALL NARGSI(4)
CALL GPLOT(Y,X,20,1)
CALL SETNAM('%XUWIND',100.)
CALL SETNAM('%XLWIND',50.)
CALL SETNAM('HISTYP',4.)
CALL SETLAB('XLABEL','HISTYP = 4')
CALL NARGSI(4)
CALL GPLOT(Y,X,20,1)
CALL NARGSI(1)
CALL GRAPHICS_HARDCOPY(0)
STOP
END
```

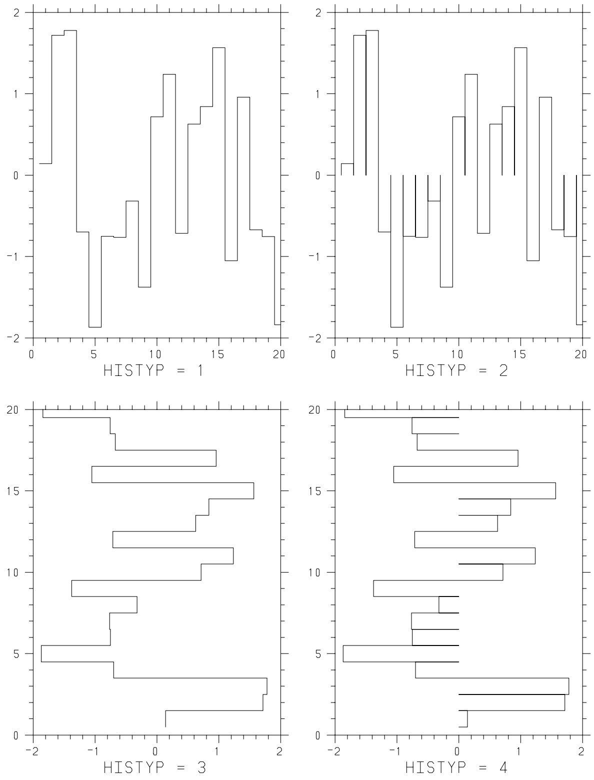


Figure 14.33: Examples of the four basic histogram types

The following program produces Figure 14.34 on page 120.

```
REAL*4 X(9), Y(9), S(9), XZ(2), YZ(2)
LOGICAL*1 P(9)
CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
CALL CLEAR_PLOT
DO I = 1, 9                  ! generate some "data"
  X(I) = FLOAT((I-1)*3)
  Y(I) = X(I)*(X(I)-20.)+50
  P(I) = 0
  S(I) = 0.8
END DO
CALL SETNAM('%YLWIND',50.)
CALL SETNAM('%XUWIND',50.)
CALL SETNAM('HISTYP',1.)     ! histogram with no tails
CALL SETLAB('XLABEL','HISTYP = 1')
CALL NARGSI(4)
CALL GPLOT(X,Y,9,1)
CALL SETNAM('%XUWIND',100.)
CALL SETNAM('%XLWIND',50.)
CALL SETNAM('MASK',-2.)
CALL SETLAB('XLABEL','Narrow bars')
CALL NARGSI(6)
CALL GPLOT(X,Y,9,1,P,S)
DO I = 1, 9
  P(I) = 8
  S(I) = 0.8
END DO
CALL SETNAM('%XUWIND',50.)
CALL SETNAM('%XLWIND',0.)
CALL SETNAM('%YUWIND',50.)
CALL SETNAM('%YLWIND',0.)
CALL SETNAM('HISTYP',2.)     ! histogram with tails
CALL SETNAM('MASK',-2.)
CALL SETLAB('XLABEL','Hatch pattern #8')
CALL HATCH_SCALE(8,0.02)
CALL NARGSI(6)
CALL GPLOT(X,Y,9,1,P,S)
XZ(1) = 0.0                  ! horizontal line thru (0,0)
YZ(1) = 0.0
XZ(1) = 25.0
YZ(1) = 0.0
CALL SETNAM('MASK',0.)
CALL NARGSI(4)
CALL GPLOT(YZ,XZ,2,2)
DO I = 1, 9
```

```
  P(I) = I*11
END DO
CALL SETNAM('%XUWIND',100.)
CALL SETNAM('%XLWIND',50.)
CALL SETNAM('MASK',-2.)
CALL SETLAB('XLABEL','Individual bar filling')
CALL NARGSI(6)
CALL GPLOT(X,Y,9,1,P,S)
CALL SETNAM('MASK',0.)
CALL NARGSI(4)
CALL GPLOT(YZ,XZ,2,2)
CALL NARGSI(1)
CALL GRAPHICS_HARDCOPY(0)
STOP
END
```

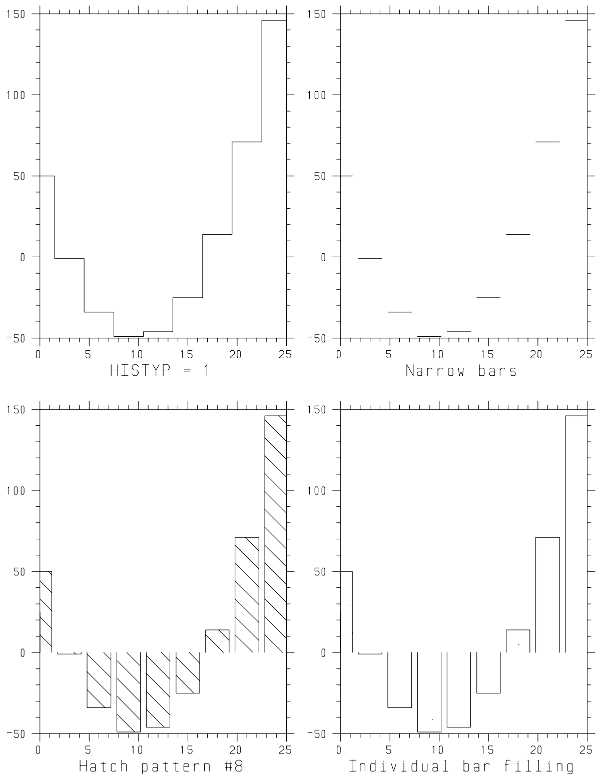


Figure 14.34: Examples of specialized histogram types

Default value: `%XLWIND` $= 0$

`XLWIND` is the left edge of the **GPLOT** window, expressed in world coordinate units. See Figure 14.35 on page 121.

`%XLWIND` is the left edge of the window expressed as a percentage of the width of the world window, that is, $\mathtt{XLWIND} = \mathtt{XMINHP} + \mathtt{\%XLWIND} \times (\mathtt{XMAXHP} - \mathtt{XMINHP}) \div 100$.

Note: The world window, as defined in the call to `HARDCOPY_RANGE`, can be found in the `/HARDCOPYRANGE/` common block.

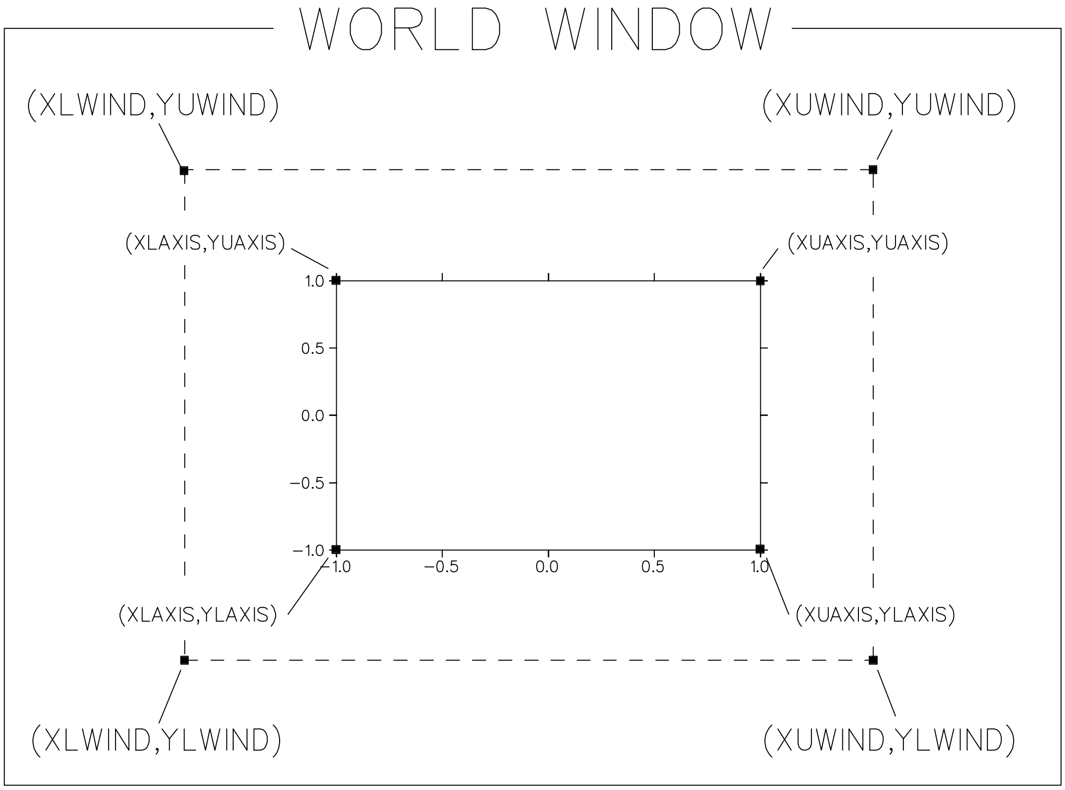


Figure 14.35: The GPLOT window and axis locations

XUWIND

Default value: `%XUWIND` $= 100$

XUWIND is the right edge of the **GPLOT** window, expressed in world coordinate units. See Figure 14.35 on page 121.

%XUWIND is the right edge of the window, expressed as a percentage of the width of the world window, that is, $\texttt{XUWIND} = \texttt{XMINHP} + \texttt{\%XUWIND} \times (\texttt{XMAXHP} - \texttt{XMINHP}) \div 100$.

YLWIND

Default value: $\texttt{\%YLWIND} = 0$

YLWIND is the lower edge of the **GPLOT** window, expressed in world coordinate units. See Figure 14.35 on page 121.

%YLWIND is the lower edge of the window, expressed as a percentage of the height of the world window, that is, $\texttt{YLWIND} = \texttt{YMINHP} + \texttt{\%YLWIND} \times (\texttt{YMAXHP} - \texttt{YMINHP}) \div 100$.

YUWIND

Default value: $\texttt{\%YUWIND} = 100$

YUWIND is the upper edge of the **GPLOT** window, expressed in world coordinate units. See Figure 14.35 on page 121.

%YUWIND is the upper edge of the window, expressed as a percentage of the height of the world window, that is, $\texttt{YUWIND} = \texttt{YMINHP} + \texttt{\%YUWIND} \times (\texttt{YMAXHP} - \texttt{YMINHP}) \div 100$.

14.3 Text keywords

CURSOR

Default value: $\texttt{CURSOR} = 1$

CURSOR controls the justification of the text that is drawn when using the TEXT keyword with the SETLAB routine. The origin of the text is always the lower left corner of the string. The justification determines where this origin is placed with respect to a reference point. Refer to Figure 14.36 on page 123 and to Table 14.37 on page 124.

`CURSOR` ≤ 0	then `XLOC` and `YLOC` will be used to determine the reference point for the text, and the justification will be determined by $\vert$`CURSOR`$\vert$.
`CURSOR` > 0	the user selects a reference point with the graphics cursor. The justification of the text is selected interactively. Refer to Table 14.38 on page 125.

The value of `CURSOR` will be updated to the value corresponding to the justification that was chosen. For example, if `CURSOR` is currently 5 and the user choses the justification code `CV`, then the value of `CURSOR` will be changed to 6. The values of `%XLOC` and `%YLOC` will also be updated to the graphics cursor position that was chosen.

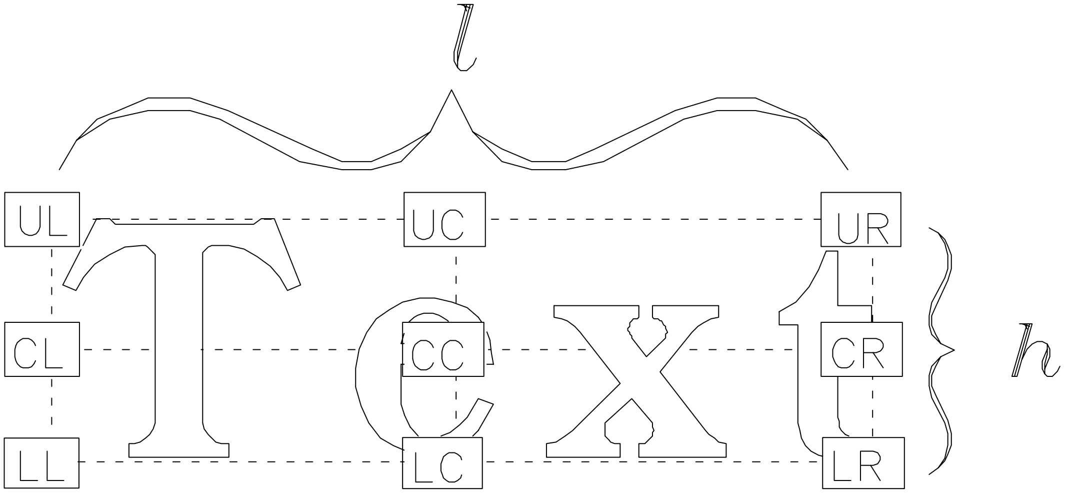


Figure 14.36: Text extent rectangle with two-character justification codes

TXTANG

Default value: `TXTANG` $= 0$

`TXTANG` controls the angle at which text will be drawn, when using the `TEXT` keyword with the `SETLAB` routine.

`TXTANG` is the angle, in degrees, measured counterclockwise, between the base line of the text string and a horizontal line. The value of `TXTANG` will be ignored if `CURSOR`$\vert = 4,\ 5,$ or 6.

TXTHIT

CURSOR	*Justification*			
	(x_0, y_0) = origin point of string (lower left corner) (x_{ref}, y_{ref}) = reference point (x_{ref} = XLOC, y_{ref} = YLOC) h = maximum height of string (default = TXTHIT) l = length of string in world coordinates a = angle of string in degrees (default = TXTANG)			
> 0.0	See the following table			
0.0 or -1.0	LL	$x_0 = x_{ref}$	$y_0 = y_{ref}$	
-2.0	LC	$x_0 = x_{ref} + l/2$	$y_0 = y_{ref}$	
-3.0	LR	$x_0 = x_{ref} + l$	$y_0 = y_{ref}$	
-4.0	LU	$x_0 = x_{ref}$	$y_0 = y_{ref}$	$a = 90°$
-5.0	LD	$x_0 = x_{ref}$	$y_0 = y_{ref}$	$a = 270°$
-6.0	CV	$x_0 = x_{ref} + l/2$	$y_0 = y_{ref}$	$a = 90°$
-7.0	CL	$x_0 = x_{ref}$	$y_0 = y_{ref} + h/2$	
-8.0	UL	$x_0 = x_{ref}$	$y_0 = y_{ref} + h$	
-9.0	CC	$x_0 = x_{ref} + l/2$	$y_0 = y_{ref} + h/2$	
-10.0	UC	$x_0 = x_{ref} + l/2$	$y_0 = y_{ref} + h$	
-11.0	CR	$x_0 = x_{ref} + l$	$y_0 = y_{ref} + h/2$	
-12.0	UR	$x_0 = x_{ref} + l$	$y_0 = y_{ref} + h$	

Table 14.37: Text justification interaction with CURSOR

<table>
<tr><th><i>key typed</i></th><th>justification with respect to reference point at crosshair location when CURSOR > 0.0</th></tr>
<tr><td>M</td><td>display the menu</td></tr>
<tr><td>Q</td><td>quit, do not draw any text</td></tr>
<tr><td>/</td><td>clear the alpha–numeric terminal screen, but not the graphics</td></tr>
<tr><td>J</td><td>a menu of justifications will be displayed. To draw the text string, choose one of the following two–character codes and type the RETURN key
<table>
<tr><th>Two character code</th><th colspan="2">Justification chosen</th></tr>
<tr><td>LL</td><td>lower left</td><td>(CURSOR set to 1)</td></tr>
<tr><td>CL</td><td>centre left</td><td>(CURSOR set to 7)</td></tr>
<tr><td>UL</td><td>upper left</td><td>(CURSOR set to 8)</td></tr>
<tr><td>LC</td><td>lower centre</td><td>(CURSOR set to 2)</td></tr>
<tr><td>CC</td><td>center centre</td><td>(CURSOR set to 9)</td></tr>
<tr><td>UC</td><td>upper centre</td><td>(CURSOR set to 10)</td></tr>
<tr><td>LR</td><td>lower right</td><td>(CURSOR set to 3)</td></tr>
<tr><td>CR</td><td>centre right</td><td>(CURSOR set to 11)</td></tr>
<tr><td>UR</td><td>upper right</td><td>(CURSOR set to 12)</td></tr>
<tr><td>LU</td><td>lower left at 90°</td><td>(CURSOR set to 4)</td></tr>
<tr><td>LD</td><td>lower left at 270°</td><td>(CURSOR set to 5)</td></tr>
<tr><td>CV</td><td>lower centre at 90°</td><td>(CURSOR set to 6)</td></tr>
</table></td></tr>
<tr><td>L</td><td>lower left (LL)</td></tr>
<tr><td>C</td><td>lower centre (LC)</td></tr>
<tr><td>R</td><td>lower right (LR)</td></tr>
<tr><td>U</td><td>lower left with an angle of 90° (LU)</td></tr>
<tr><td>D</td><td>lower left with an angle of 270° (LD)</td></tr>
<tr><td>V</td><td>lower centre with an angle of 90° (CV)</td></tr>
<tr><td>X</td><td>lower left (LL); using the y location selected by the crosshair and the x location that is stored in XLOC</td></tr>
<tr><td>Y</td><td>lower left (LL); using the x location selected by the crosshair and the y location that is stored in YLOC</td></tr>
<tr><td>other</td><td>use current value of CURSOR for justification
use the crosshair position for the reference point</td></tr>
</table>

Table 14.38: Text menu and justification

Default value: %TXTHIT $= 3$

TXTHIT controls the height of text to be drawn, when using the TEXT keyword with the SETLAB routine. %TXTHIT is a percentage of the height of the **GPLOT** window, that is, TXTHIT $=$ %TXTHIT $\times$ (YUWIND $-$ YLWIND) $\div 100$

XLOC

Default value: %XLOC $= 50$

XLOC is the horizontal reference position of a text string drawn using the TEXT keyword used with the SETLAB routine. This reference point is used for text justification. See the explanation of CURSOR for how XLOC will be used. XLOC will be used if CURSOR < 0, or if the X key is typed when drawing text in interactive mode. The value of %XLOC is updated after each call to SETLAB with the TEXT keyword. %XLOC is a percentage of the width of the window, that is, XLOC $=$ XLWIND $+$ %XLOC $\times$ (XUWIND $-$ XLWIND) $\div 100$

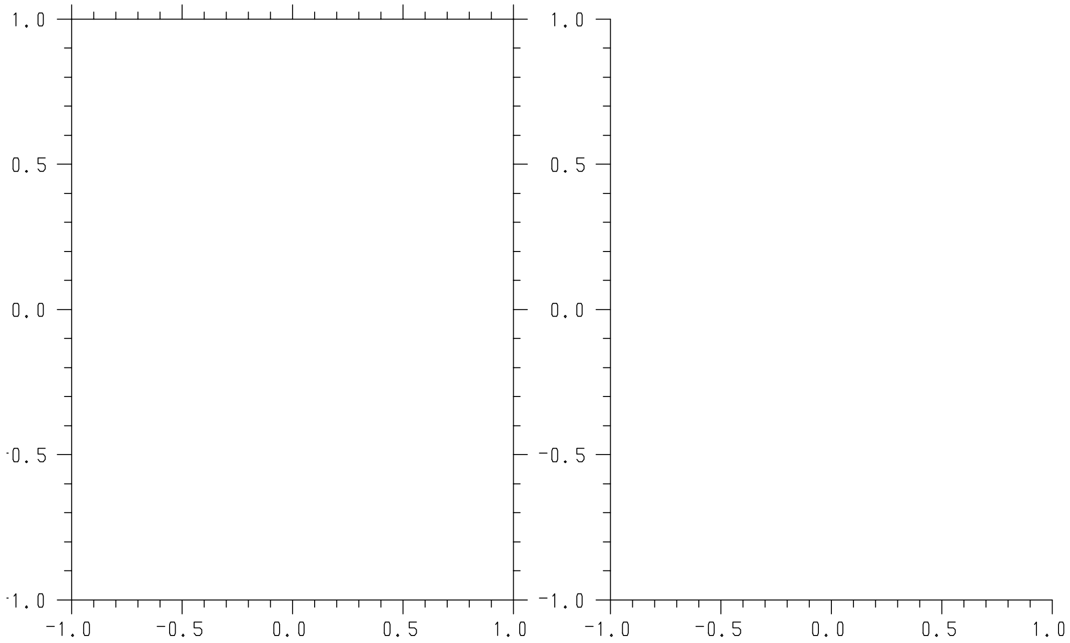
YLOC

Default value: %YLOC $= 50$

YLOC is the vertical reference position of a text string drawn using the TEXT keyword used with the SETLAB routine. This reference point is used for text justification. See the explanation of CURSOR for how YLOC will be used. YLOC will be used if CURSOR < 0, or if the Y key is typed when drawing text in interactive mode. The value of %YLOC is updated after each call to SETLAB with the TEXT keyword. %YLOC is a percentage of the height of the window, that is, YLOC $=$ YLWIND $+$ %YLOC $\times$ (YUWIND $-$ YLWIND) $\div 100$

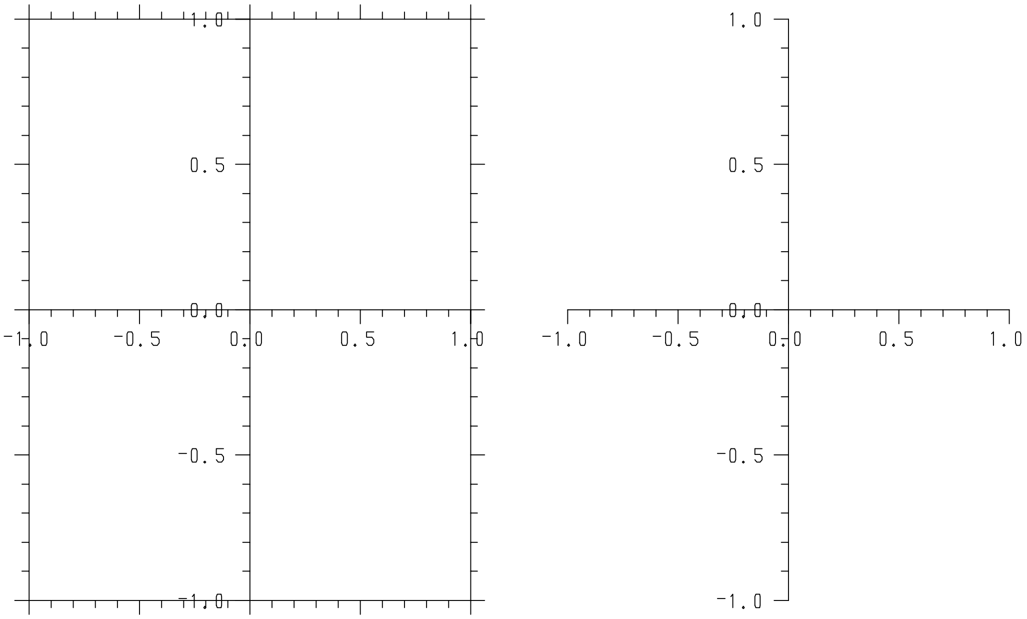
14.4 Axis box keywords

BOX



using the defaults

BOX = 0



XCROSS=1, YCROSS=1, BOX=1

XCROSS=1, YCROSS=1, BOX=0

Figure 14.37: Examples of the graph box

Default value: $\texttt{BOX} = 1$

`BOX` controls whether or not an axis box is placed around the graph. See Figure 14.37 on page 127.

$\texttt{BOX} = 0$ do not draw an axis box
$\texttt{BOX} \neq 0$ draw an axis box

XLAXIS

Default value: $\texttt{\%XLAXIS} = 12$

`XLAXIS` controls the position of the left, or lower, end of the x-axis. See Figure 14.35 on page 121. This is also the horizontal coordinate of the lower left hand corner of the axis box, if $\texttt{BOX} = 1$. `%XLAXIS` is a percentage of the width of the window, that is, $\texttt{XLAXIS} = \texttt{XLWIND} + \texttt{\%XLAXIS} \times (\texttt{XUWIND} - \texttt{XLWIND}) \div 100$

XUAXIS

Default value: $\texttt{\%XUAXIS} = 94$

`XUAXIS` controls the position of the right, or upper, end of the x-axis. See Figure 14.35 on page 121. This is also the horizontal coordinate of the upper right hand corner of the graph box, if $\texttt{BOX} = 1$. `%XUAXIS` is a percentage of the width of the window, that is, $\texttt{XUAXIS} = \texttt{XLWIND} + \texttt{\%XUAXIS} \times (\texttt{XUWIND} - \texttt{XLWIND}) \div 100$

XAXISA

Default value: $\texttt{XAXISA} = 0$

`XAXISA` is the angle, in degrees, measured counterclockwise, between a horizontal line and the x-axis. See Figure 14.39 on page 139.

YLAXIS

Default value: %YLAXIS $= 12$

YLAXIS controls the position of the bottom, or lower, end of the y-axis. See Figure 14.35 on page 121. This is also the vertical coordinate of the lower left hand corner of the graph box, if BOX $= 1$. %YLAXIS is a percentage of the height of the window, that is, YLAXIS $=$ %YLAXIS $\times$ (YUWIND $-$ YLWIND) $\div 100$

YUAXIS

Default value: %YUAXIS $= 94$

YUAXIS controls the position of the upper end of the y-axis. See Figure 14.35 on page 121. This is also the vertical coordinate of the upper right hand corner of the graph box, if BOX $= 1$. %YUAXIS is a percentage of the height of the window, that is, YUAXIS $=$ %YUAXIS $\times$ (YUWIND $-$ YLWIND) $\div 100$

YAXISA

Default value: YAXISA $= 90$

YAXISA is the angle, in degrees, measured counterclockwise, between a horizontal line and the y-axis. See Figure 14.42 on page 154.

BOTNUM

Default value: BOTNUM $= 0$

BOTNUM controls the height of the numbers on the bottom edge of the box. |BOTNUM| is the ratio of the height of the numbers on the bottom edge of the box to XNUMSZ, the height of the numbers on the x-axis. The height of these numbers will be |BOTNUM| $\times$ XNUMSZ.

BOTNUM is ignored if BOX $= 0$ or if the bottom edge of the axis box overlaps the x-axis.

`BOTNUM` $= 0$	no numbers on the bottom edge of the box
`BOTNUM` > 0	numbers on bottom edge of the box on same side as x-axis numbers
`BOTNUM` < 0	numbers on bottom edge of the box on opposite side as x-axis numbers

BOTTIC

Default value: `BOTTIC` $= 1$

`BOTTIC` controls the lengths of the large and short tic marks on the bottom edge of the box. |`BOTTIC`| is the ratio of the lengths of the tic marks on the bottom edge of the box to the lengths of the tic marks on the x-axis, where `XTICL` is the length of the large x-axis tic marks and `XTICS` is the length of the short x-axis tic marks. The large tic marks on the bottom of the box will have a length of |`BOTTIC`| × `XTICL` and the short tic mark length will be |`BOTTIC`| × `XTICS`.

`BOTTIC` is ignored if `BOX` $= 0$ or if the bottom edge of the axis box overlaps the x-axis.

`BOTNUM` $= 0$	no tic marks on the bottom edge of the box
`BOTNUM` > 0	tic marks on bottom edge of the box on the same side as the x-axis tic marks
`BOTNUM` < 0	tic marks on bottom edge of the box on the opposite side as the x-axis tic marks

RITNUM

Default value: `RITNUM` $= 0$

`RITNUM` controls the height of the numbers on the right edge of the box. |`RITNUM`| is the ratio of the height of the numbers on the right edge of the box to `YNUMSZ`, the height of the numbers on the y-axis. This number height will be |`RITNUM`| × `YNUMSZ`.

`RITNUM` is ignored if `BOX` $= 0$ or if the right edge of the axis box overlaps the y-axis.

`RITNUM` $= 0$	no numbers on the right edge of the box
`RITNUM` > 0	numbers on right edge of the box on the same side as y-axis numbers
`RITNUM` < 0	numbers on right edge of the box on the opposite side as y-axis numbers

RITTIC

Default value: `RITTIC` $= -1$

`RITTIC` controls the lengths of the large and short tic marks on the right edge of the box. $|$`RITTIC`$|$ is the ratio of the lengths of the tic marks on the right edge of the box to the lengths of the tic marks on the y-axis. `YTICL` is the length of the large y-axis tic marks and `YTICS` is the length of the short y-axis tic marks. The large tic marks on the right edge of the box will have a length of $|$`RITTIC`$|\times$`YTICL` and the short tic mark length will be $|$`RITTIC`$|\times$`YTICS`.

`RITTIC` is ignored if `BOX` $= 0.0$ or if the right edge of the axis box overlaps the y-axis.

`RITTIC` $= 0$	no tic marks on the right edge of the box
`RITTIC` > 0	tic marks on right edge of the box on same side as y-axis tic marks
`RITTIC` < 0	tic marks on right edge of the box on opposite side as y-axis large tic marks

TOPNUM

Default value: `TOPNUM` $= 0$

`TOPNUM` controls the height of the numbers on the top edge of the box. $|$`TOPNUM`$|$ is the ratio of the height of the numbers on the top edge of the box to `XNUMSZ`, the height of the numbers on the x-axis. The height of these numbers will be $|$`TOPNUM`$|$ $\times$ `XNUMSZ`.

`TOPNUM` is ignored if `BOX` $= 0$ or if the top edge of the axis box overlaps the x-axis.

`TOPNUM` $= 0$	no numbers on the top edge of the box
`TOPNUM` > 0	numbers on top edge of the box on the same side as x-axis numbers
`TOPNUM` < 0	numbers on top edge of the box on the opposite side as x-axis numbers

TOPTIC

Default value: `TOPTIC` $= -1$

`TOPTIC` controls the lengths of the large and short tic marks on the top edge of the box. |`TOPTIC`| is the ratio of the lengths of the tic marks on the top edge of the box to the lengths of the tic marks on the x-axis, where `XTICL` is the length of the large x-axis tic marks and `XTICS` is the length of the short x-axis tic marks. The large tic marks on the top edge of the box will have a length of |`TOPTIC`| × `XTICL` and the short tic mark length will be |`TOPTIC`| × `XTICS`.

`TOPTIC` is ignored if `BOX` $= 0$ or if the top edge of the axis box overlaps the x-axis.

`TOPTIC` $= 0$	no tic marks on the top edge of the box
`TOPTIC` > 0	tic marks on top edge of the box on the same side as x-axis large tic marks
`TOPTIC` < 0	tic marks on top edge of the box on the opposite side as x-axis large tic marks

LEFNUM

Default value: `LEFNUM` $= 0$

`LEFNUM` controls the height of the numbers on the left edge of the box. |`LEFNUM`| is the ratio of the height of the numbers on the left edge of the box to `YNUMSZ`, the height of the numbers on the y-axis. The height of these numbers will be |`LEFNUM`| × `YNUMSZ`.

`LEFNUM` is ignored if `BOX` $= 0$ or if the left edge of the axis box overlaps the y-axis.

`LEFNUM` $= 0$	no numbers on the left edge of the box
`LEFNUM` > 0	numbers on left edge of the box on the same side as y-axis numbers
`LEFNUM` < 0	numbers on left edge of the box on the opposite side as y-axis numbers

LEFTIC

Default value: `LEFTIC` $= 1$

`LEFTIC` controls the lengths of the large and short tic marks on the bottom edge of the box. |`LEFTIC`| is the ratio of the lengths of the tic marks on the left edge of the box to the lengths of the tic marks on the y-axis, where `YTICL` is the length of the large y-axis tic marks and `YTICS` is the length of the short y-axis tic marks. The large tic marks on the left edge of the box will have a length of |`LEFTIC`| × `YTICL` and the short tic mark length will be |`LEFTIC`| × `YTICS`.

`LEFTIC` is ignored if `BOX` $= 0$ or if the left edge of the axis box overlaps the y-axis.

`LEFTIC` $= 0$	no tic marks on the left edge of the box
`LEFTIC` > 0	tic marks on left edge of the box on the same side as y-axis tic marks
`LEFTIC` < 0	tic marks on left edge of the box on the opposite side as y-axis tic marks

14.5 Plotting symbol keywords

MASK

Default value: `MASK` $= 0$

`MASK`, in conjunction with the `PMODE` keyword, controls the way in which the plotting symbols will be drawn at the data points. Table 14.39 on page 134 describes how `MASK` interacts with `PMODE` when `HISTYP` $= 0$, while Table 14.40 on page 134 describes how `MASK` is interpreted when `HISTYP` > 0. In the following discussion, $pangl$, $pcolr$, $psize$, and $pchar$ refer to the optional arguments for the `GPLOT` routine.

- If `MASK` $= -4$ but $pangl$ is not supplied, `MASK` will be set to -3.
- If `MASK` $= -3$ but $pcolr$ is not supplied, `MASK` will be set to -2.
- If `MASK` $= -2$ but $psize$ is not supplied, `MASK` will be set to -1.
- If `MASK` $= -1$ but $pchar$ is not supplied, `MASK` will be set to 0.

This checking and possible resetting of `MASK` is done sequentially *each* time the `GPLOT` routine is called. For example, consider the following program segment:

```
CALL SETNAM('MASK',-4.)
CALL NARGSI(4)
CALL GPLOT(X,Y,NPTS,IAXIS)
XMASK = GETNAM('MASK')
```

The value of `XMASK` will be 0.

		plotting symbol at (x_i, y_i)			
		symbol type	*size*	*colour*	*angle*
`PMODE` $= 0$	`MASK` ignored	no symbol			
`PMODE` $\neq 0$	`MASK` $= m > 1$	`CHAR(2:2)` if $i \bmod m = 0$ `CHAR(1:1)` if $i \bmod m \neq 0$	`CHARSZ`	current	`CHARA`
	`MASK` $= 1$	`CHAR(1:1)` for $1 \leq i \leq$ `NPT`	`CHARSZ`	current	`CHARA`
	`MASK` $= 0$	no symbol			
	`MASK` $= -1$	$pchar_i$	`CHARSZ`	current	`CHARA`
	`MASK` $= -2$	$pchar_i$	$psize_i$	current	`CHARA`
	`MASK` $= -3$	$pchar_i$	$psize_i$	$pcolr_i$	`CHARA`
	`MASK` $= -4$	$pchar_i$	$psize_i$	$pcolr_i$	`PANGL`$_i$

Table 14.39: The relationship of `MASK` and `PMODE` to the plotting symbol when `HISTYP` $= 0$

	histogram bar at (x_i, y_i)		
	hatch pattern	*bar width scale factor*	*colour*
`MASK` ≥ 0	none	1	current
`MASK` $= -1$	$pchar_i$	1	current
`MASK` $= -2$	$pchar_i$	$psize_i$	current
`MASK` ≤ -3	$pchar_i$	$psize_i$	$pcolr_i$

Table 14.40: The relationship of `MASK` to the histogram bar when `HISTYP` > 0

PMODE

Default value: `PMODE` $= 1$

`PMODE` controls whether or not the plotting symbols drawn at the data points will be joined with line segments. `PMODE` is ignored if `MASK` $= 0$. When the the optional argument $pchar$ is supplied to `GPLOT` routine, the sign of `PMODE` is multiplied by the sign of $pchar_i$ to determine whether the data points are connected or not.

$\texttt{PMODE} = 1$	plotting symbols joined with line segments
$\texttt{PMODE} = 0$	line segments joining the data points, no plotting symbols
$\texttt{PMODE} = -1$	plotting symbols, not joined with line segments

CHARA

Default value: CHARA = XAXISA

CHARA is the angle, in degrees, measured counterclockwise, between the a horizontal line and a base line through the plotting symbol. CHARA will be ignored if the optional argument $pangl$ is included in the call to the GPLOT routine.

If CHARA is set as a percentage, then CHARA is set to XAXISA, and the actual value to which %CHARA has been set will be ignored. This allows the user to change the angle of the x-axis and keep the plotting symbols base line parallel to the x-axis. By default, CHARA is set as a percentage.

CHARSZ

Default value: %CHARSZ $= 1$

CHARSZ is the size of the plotting symbols that are drawn at the data points. %CHARSZ is a percentage of the height of the **GPLOT** window, that is, $\texttt{CHARSZ} = \texttt{\%CHARSZ} \times (\texttt{YUWIND} - \texttt{YLWIND}) \div 100$.

CHARSZ will be the base size of the plotting symbols if the optional argument $psize$ is included in the call to the GPLOT routine. The size of the plotting symbol drawn at the i^{th} data point will be $psize_i \times \texttt{CHARSZ}$.

ERRBAR

Default value: ERRBAR $= 0$

ERRBAR controls the type of error bars to be used at the data points. See Figure 14.38 on page 136.

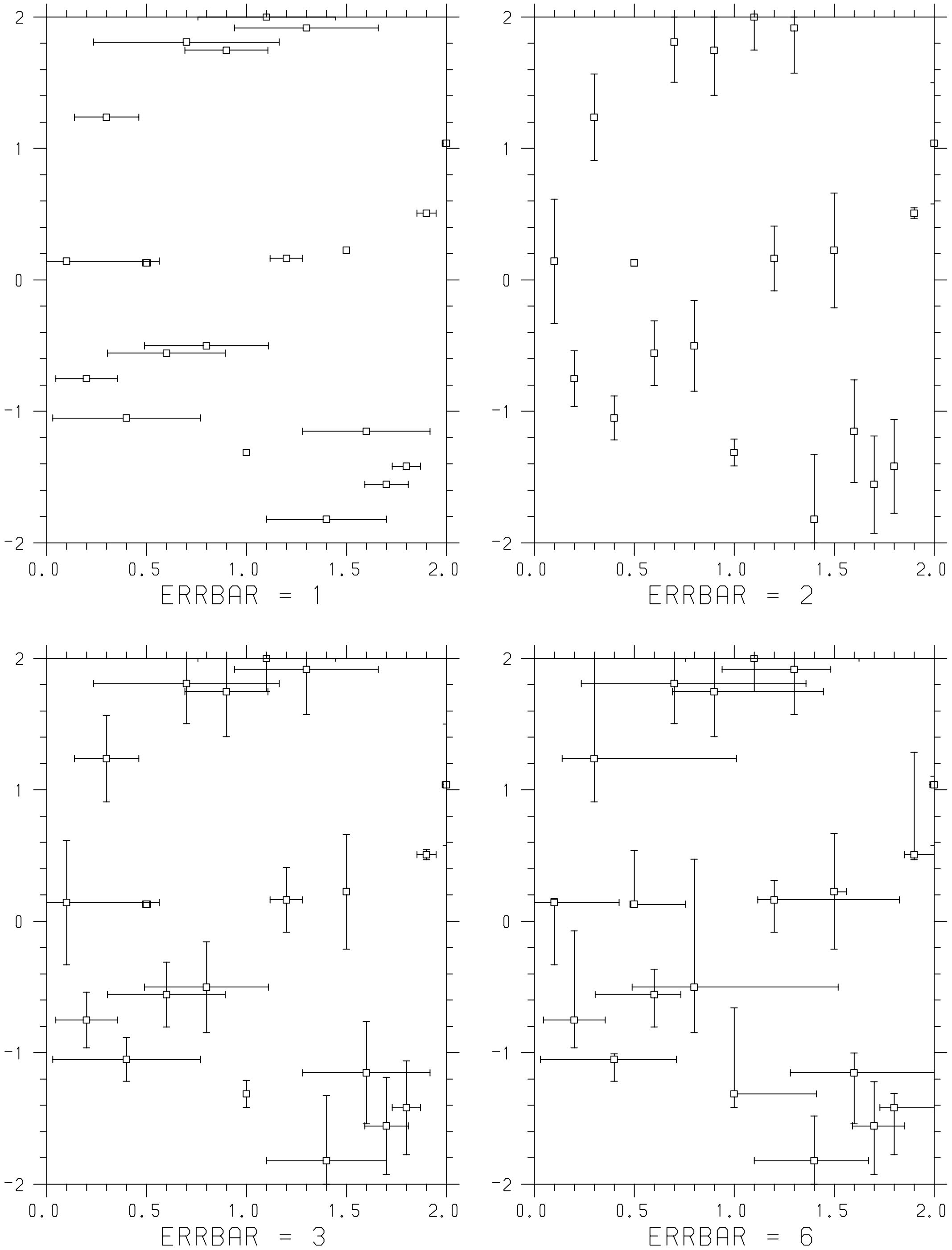


Figure 14.38: Examples of errbars

`ERRBAR` $= 0$	no error bars will be drawn. The data arrays x and y, passed to the `GPLOT` routine, are assumed to be singly dimensioned, and npt is assumed to be a single number.
`ERRBAR` $= 1$	symmetric x-error bars only. The data array x must have two dimensions, with the second dimension equal to two (2). The x-error at the i_{th} point is assumed from $x(i,1) - x(i,2)$ to $x(i,1) + x(i,2)$. The number of points to plot is assumed to be $npt(1)$, but the first dimension of x, as declared in the calling program, should be $npt(2)$.
`ERRBAR` $= 2$	symmetric y-error bars only. The data array y must have two dimensions, with the second dimension equal to two (2). The y-error at the i_{th} point is assumed to be $y(i,1) - y(i,2)$ and $y(i,1) + y(i,2)$. The number of points to plot is assumed to be $npt(1)$, but the first dimension of y, as declared in the calling program, should be $npt(2)$.
`ERRBAR` $= 3$	symmetric x and y error bars. Both arrays x and y must have two dimensions, with the second dimension equal to two (2). The x and y errors at the i_{th} point are assumed to be $x(i,1) - x(i,2)$, $x(i,1) + x(i,2)$; $y(i,1) - y(i,2)$, and $y(i,1) + y(i,2)$. The number of points to plot is assumed to be $npt(1)$, but the first dimension of x and y, as declared in the calling program, should be $npt(2)$.
`ERRBAR` $= 4$	assymmetic x-error bars only. The data array x must have two dimensions, with the second dimension equal to three (3). The x-error at the i_{th} point is assumed to be $x(i,1) - x(i,2)$, and $x(i,1) + x(i,3)$. The number of points to plot is assumed to be $npt(1)$, but the first dimension of x, as declared in the calling program, should be $npt(2)$.
`ERRBAR` $= 5$	assymmetric y-error bars only. The data array y must have three dimensions (3), with the second dimension equal to three (3). The y-error at the i_{th} point is assumed to be $y(i,1) - y(i,2)$, and $y(i,1) + y(i,3)$. The number of points to plot is assumed to be $npt(1)$, but the first dimension of y, as declared in the calling program, should be $npt(2)$.
`ERRBAR` $= 6$	assymmetric x and y error bars. The data arrays x and y must have three dimensions, with the second dimension equal to three (3). The x and y error at the i_{th} point is assumed to be $x(i,1) - x(i,2)$, and $x(i,1) + x(i,3)$; $y(i,1) - y(i,2)$, and $y(i,1) + y(i,3)$. The number of points to plot is assumed to be $npt(1)$, but the first dimension of x and y, as declared in the calling program, should be $npt(2)$.

Table 14.41: Using `ERRBAR` to draw error bars

ERRBAR $= 0$	`REAL*4 X(10), Y(10)`
	`INTEGER*4 NPT`

ERRBAR $= 1$	`REAL*4 X(10,2), Y(10)`
	`INTEGER*4 NPT(2)`
	`NPT(2) = 10`

ERRBAR $= 2$	`REAL*4 X(10), Y(10,2)`
	`INTEGER*4 NPT(2)`
	`NPT(2) = 10`

ERRBAR $= 3$	`REAL*4 X(10,2), Y(10,2)`
	`INTEGER*4 NPT(2)`
	`NPT(2) = 10`

ERRBAR $= 4$	`REAL*4 X(10,3), Y(10)`
	`INTEGER*4 NPT(2)`
	`NPT(2) = 10`

ERRBAR $= 5$	`REAL*4 X(10), Y(10,3)`
	`INTEGER*4 NPT(2)`
	`NPT(2) = 10`

ERRBAR $= 6$	`REAL*4 X(10,3), Y(10,3)`
	`INTEGER*4 NPT(2)`
	`NPT(2) = 10`

Table 14.42: ERRBAR examples

The length of the 'foot' of the error bar will be `CHARSZ`. If the optional argument $psize$ is included in the call to the `GPLOT` routine, the size of the 'foot' of the error bar drawn at the i^{th} data point will be $psize_i \times$ `CHARSZ`. If one of the symmetric plotting symbols, for example, the 'box' or the 'circle', is being used, the error bar is hidden behind the plotting symbol.

14.6 x-axis keywords

Figure 14.39 on page 139 illustrates the definitions of some of the x-axis keywords.

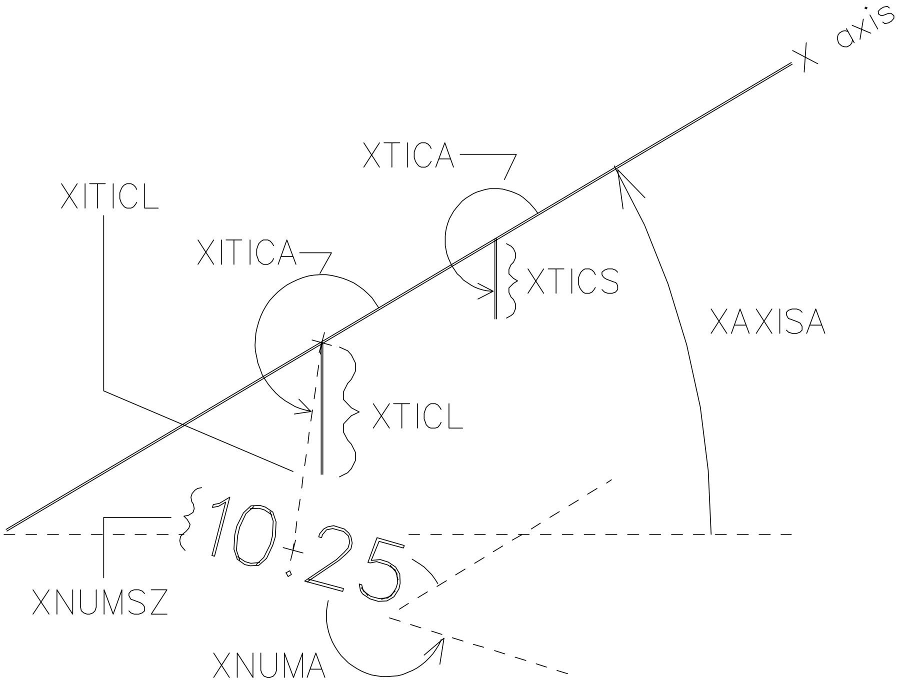


Figure 14.39: Examples of some x-axis characteristics

XAXIS

Default value: `XAXIS` $= 1$

`XAXIS` controls whether or not the x-axis is drawn.

`XAXIS` $= 0$	do not draw the x-axis
`XAXIS` $\neq 0$	draw the x-axis

If `BOX` $\neq 0$, then the bottom of the axis box will be drawn, even if `XAXIS` $= 0$.

XLABSZ

Default value: `%XLABSZ` $= 3$

`XLABSZ` controls the height of the automatic text label for the x-axis, set by the `XLABEL` keyword using the `SETLAB` routine. `%XLABSZ` is a percentage of the height of the **GPLOT** window, that is, `XLABSZ` $=$ `%XLABSZ` $\times$ (`YUWIND` $-$ `YLWIND`) $\div 100$

XLOG

Default value: `XLOG` $= 0$

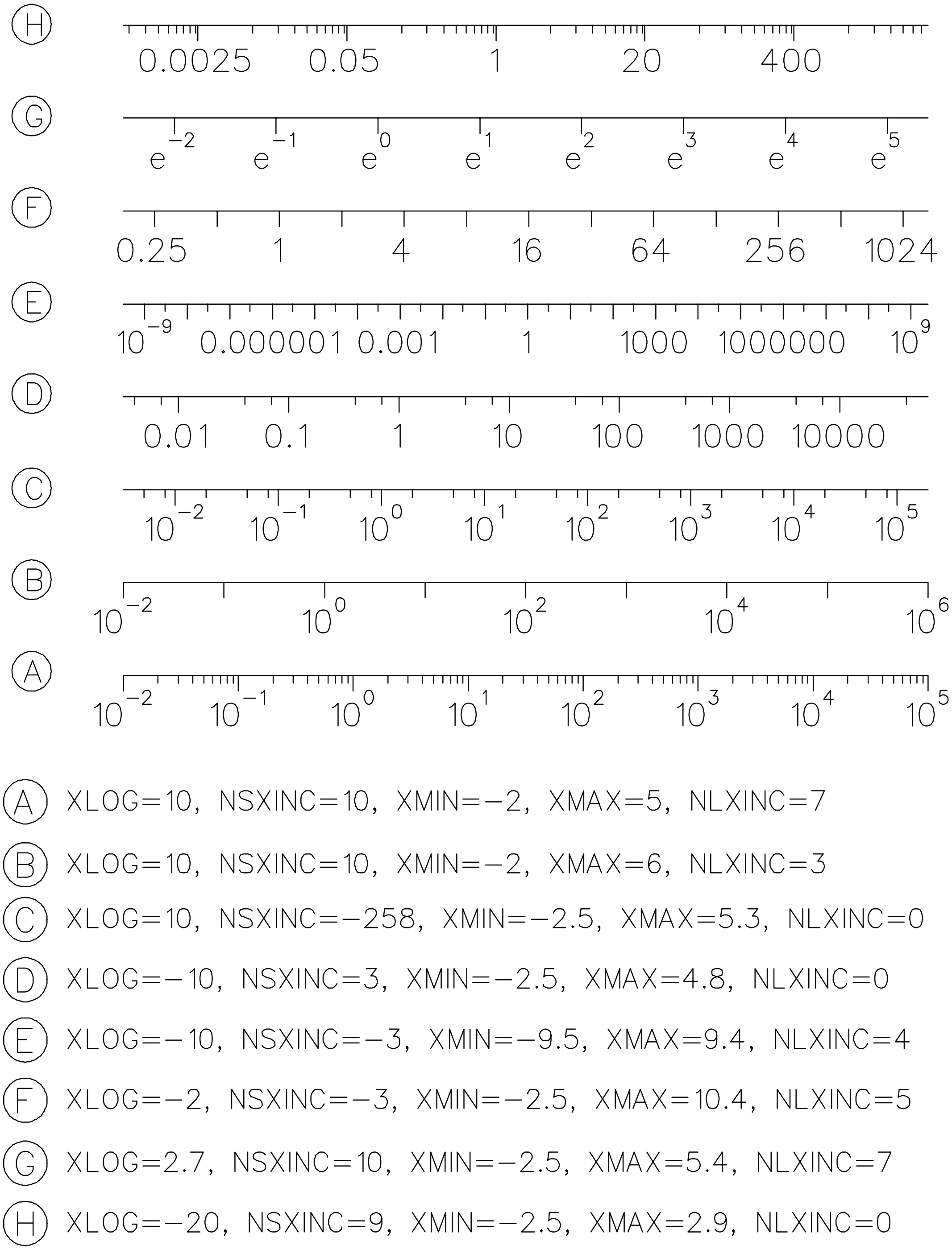
`XLOG` determines whether the x-axis is to be linear or logarithmic. See Figure 14.40 on page 141 for examples of various logarithmic axes.

`XLOG` > 1	the x-axis will have a logarithmic scale. The numbers labeling the x-axis will be in exponent form, for example, 10^1 10^2 10^3 10^4.
$-1 \leq$ `XLOG` ≤ 1	the x-axis will have a linear scale
`XLOG` < -1	the x-axis will have a logarithmic scale. The numbers labeling the x-axis will be in full digit form, for example, 10 100 1000 10000.

Suppose that $|$`XLOG`$| > 1$. The base will be the *integer part* of $|$`XLOG`$|$, except for the special case: $1.05 \times e >$ `XLOG` $> 0.95 \times e$, where e is the base of the natural logarithms, $e \approx 2.718281828$, in which case the base will be e. `XMIN` and `XMAX` will be the *exponents* for the minimum and maximum values on the x-axis. The maximum value displayed on the x-axis is $|\mathtt{XLOG}|^{\mathtt{XMAX}}$ and the minimum value displayed on the x-axis is $|\mathtt{XLOG}|^{\mathtt{XMIN}}$. Integer exponents are always used for the axis numbering, so the x-axis may not begin and/or end on a large tic mark.

If $|$`XLOG`$| = 10$, the small tic marks on the x-axis can be specified exactly with `NSXINC`. If small tic marks are desired at the locations $j_m \times 10^n$, where $2 \leq j_m \leq 9$, then set `NSXINC` $= -j_1 \ldots j_m$. For example, if you want small tic marks at 2×10^n, 4×10^n, 5×10^n, and 8×10^n, then set `NSXINC` $= -2458$.

If $|$`XLOG`$| \leq 1$, then the x-axis will be linear. `XMIN` and `XMAX` will be the actual minimum and maximum values for the x-axis.



Ⓐ XLOG=10, NSXINC=10, XMIN=−2, XMAX=5, NLXINC=7

Ⓑ XLOG=10, NSXINC=10, XMIN=−2, XMAX=6, NLXINC=3

Ⓒ XLOG=10, NSXINC=−258, XMIN=−2.5, XMAX=5.3, NLXINC=0

Ⓓ XLOG=−10, NSXINC=3, XMIN=−2.5, XMAX=4.8, NLXINC=0

Ⓔ XLOG=−10, NSXINC=−3, XMIN=−9.5, XMAX=9.4, NLXINC=4

Ⓕ XLOG=−2, NSXINC=−3, XMIN=−2.5, XMAX=10.4, NLXINC=5

Ⓖ XLOG=2.7, NSXINC=10, XMIN=−2.5, XMAX=5.4, NLXINC=7

Ⓗ XLOG=−20, NSXINC=9, XMIN=−2.5, XMAX=2.9, NLXINC=0

Figure 14.40: Logarithmic x-axis examples

NXGRID

Default value: `NXGRID` $= 0$

`NXGRID` controls the number of grid lines parallel to the y-axis. `NXGRID` specifies the number of large x-axis tic marks between each grid line.

`NXGRID` > 0	grid lines parallel to the y-axis at every `NXGRID` large tic mark, starting with the first
`NXGRID` $= 0$	suppress all grid lines parallel to the y-axis
`NXGRID` $= -1$	grid lines parallel to the y-axis at every x-axis tic mark, large and small. This option applies only if the axes are orthogonal.

XCROSS

Default value: `XCROSS` $= 0$

`XCROSS` controls where the y-axis is to cross the x-axis. The axes always cross at a large tic mark.

`XCROSS` $= 0$	the y-axis will cross the x-axis at `XMIN`
`XCROSS` $\neq 0$	the y-axis will cross the x-axis at the large tic mark closest to $x = 0$

XZERO

Default value: `XZERO` $= 0$

`XZERO` controls whether or not to force zero to be displayed on the x-axis. `XZERO` is set to zero (0) after each graph is drawn.

`XZERO` $\neq 0$	if `XMIN` ≥ 0 then set `XMIN` to zero, or, if `XMAX` ≤ 0 then set `XMAX` to zero
`XZERO` $= 0$	do not force zero to be displayed on the x-axis

XTICTP

Default value: `XTICTP` $= 1$

`XTICTP` controls the type of tic marks to place on the x-axis.

`XTICTP` $= 2$	tic marks on both sides of the x-axis
`XTICTP` $\neq 2$	tic marks on only one side of the x-axis. `XTICA` controls the side on which the tic marks will appear

XTICA

Default value: `XTICA` $=$ `YAXISA` $-$ `XAXISA` $+ 180°$

`XTICA` is the angle, in degrees, measured counterclockwise, between a horizontal line and a tic mark, both large and small, on the x-axis. See Figure 14.39 on page 139. By default, `XTICA` is set as a percentage. If `XTICA` is set as a percentage, then `XTICA` is set to `YAXISA`$-$`XAXISA`$+180$. The actual value of `%XTICA` will be ignored. This allows the user to change the angle of the axes and keep the x-axis tic marks parallel to the y-axis.

NLXINC

Default value: `NLXINC` $= 5$

`NLXINC` controls the number of large tic marks to be displayed on the x-axis.

$\texttt{NLXINC} = -1000$	drop the first and the last numbers on the x-axis and determine the number of large x-axis tic marks
$\texttt{NLXINC} < 0$	drop the first and the last numbers on the x-axis. The number of large tic marks on the x-axis will be $\lvert\texttt{NLXINC}\rvert + 1$, unless $\lvert\texttt{XLOG}\rvert > 1$, in which case the number of tic marks will be determined to avoid fractional powers
$\texttt{NLXINC} = 0$	the number of large tic marks on the x-axis will be automatically determined
$\texttt{NLXINC} > 0$	the number of large tic marks on the x-axis will $\texttt{NLXINC} + 1$, unless $\lvert\texttt{XLOG}\rvert > 1$, in which case `NLXINC` will be determined to avoid fractional powers

XTICL

Default value: $\texttt{\%XTICL} = 2$

`XTICL` is the length of the long tic marks on the x-axis. See Figure 14.39 on page 139. `%XTICL` is a percentage of the height of the **GPLOT** window, that is, $\texttt{XTICL} = \texttt{\%XTICL} \times (\texttt{YUWIND} - \texttt{YLWIND}) \div 100$

NSXINC

Default value: $\texttt{NSXINC} = 5$

`NSXINC` controls the number of small tic marks to be displayed on the x-axis. The small tic marks appear between the large tic marks.

$\texttt{NSXINC} \le 1$	no small tic marks on the x-axis
$\texttt{NSXINC} \ge 2$	the number of small tic marks, on the x-axis, between each pair of large tic marks, will be $\texttt{NSXINC} - 1$

If $\lvert\texttt{XLOG}\rvert = 10$, the small tic marks on the x-axis can be specified exactly with `NSXINC`. If small tic marks are desired at the locations $j_m \times 10^n$, where $2 \le j_m \le 9$, then set $\texttt{NSXINC} = -j_1 \ldots j_m$. For example, if you want small tic marks at 2×10^n, 4×10^n, 5×10^n, and 8×10^n, then set $\texttt{NSXINC} = -2458$.

XTICS

Default value: `%XTICS` $= 1$

`XTICS` controls the length of the short tic marks on the x-axis. See Figure 14.39 on page 139. `%XTICS` is a percentage of the height of the **GPLOT** window, that is, `XTICS` $=$ `%XTICS` $\times$ (`YUWIND` $-$ `YLWIND`) $\div 100$

XAUTO

Default value: `XAUTO` $= 2$

`XAUTO` controls the autoscaling of the x-axis. `XAUTO` is ignored if the `GPLOT` argument $iaxis = 2$, that is, if only the data points are to be plotted and not the axes.

`XAUTO` is set to zero after each call to `GPLOT`.

XAUTO $= 3$	set XMIN and XMAX to the minimum and maximum of the data passed to the GPLOT routine in the x array. Determine XVMIN, XVMAX, and NLXINC so that 'nice' numbers will label the large tic marks on the x-axis. The x-axis might not begin and end on large, labeled, tic marks. Also determine XPOW, NXDIG and NXDEC. Do not set XPAUTO to 1 or 2 after setting XAUTO to 3
XAUTO $= 2$	determine XMIN and XMAX so that all the data points passed to GPLOT in the x array appears on the graph. Set XVMIN to XMIN and set XVMAX to XMAX so that the x-axis will begin and end on large, labeled, tic marks. Determine NLXINC so that 'nice' numbers will label the large tic marks on the x-axis. Also determine XPOW, NXDIG and NXDEC. Do not set XPAUTO to 1 or 2 after setting XAUTO to 2.
XAUTO $= 1$	determine XMIN and XMAX. Set XVMIN to XMIN and set XVMAX to XMAX so the x-axis will begin and end at large, labeled, tic marks. If NLXINC $= 0$ then determine NLXINC so that 'nice' numbers will label the large tic marks on the x-axis. Use the current values for XPOW, NXDIG and NXDEC.
XAUTO $= 0$	do not autoscale the x-axis, that is, use the current values for XMIN and XMAX. If NLXINC $= 0$, determine XVMIN, XVMAX, and NLXINC so that 'nice' numbers will label the large tic marks on the x-axis. The x-axis may not begin and end on large, labeled, tic marks.

If XMIN $=$ XMAX, then XMIN and XMAX will be determined from the x and y arrays passed to the GPLOT routine, just as if XAUTO had been set to one (1).

XMAX

Default value: XMAX $= 10$

XMAX controls the maximum value for the x-axis.

If $|\mathtt{XLOG}| > 1$, then XMAX is assumed to be the exponent for the maximum value on the x-axis. The maximum value displayed on the x-axis is $|\mathtt{XLOG}|^{\mathtt{XMAX}}$. Integer exponents are always used for the axis numbering, so the x-axis may not end on a large tic mark.

If $|\mathtt{XLOG}| \leq 1$, then XMAX is the actual maximum value for the x-axis. If XVMAX is equal to XMAX then the x-axis always ends on a large, labeled, tic mark. If XVMAX is not equal to XMAX then the x-axis will not end at a large, labeled, tic mark.

When the value of XMAX is changed, the value of XVMAX is simultaneously changed to the same value.

XVMAX

Default value: XVMAX $= 10$

XVMAX controls the virtual maximum value for the x-axis. See Figure 14.41 on page 149.

If $|\mathtt{XLOG}| > 1$, the virtual maximum is ignored.

If $|\mathtt{XLOG}| \leq 1$, then XVMAX is the virtual maximum value of the labels for the x-axis. This value will not be displayed if XVMAX is greater than XMAX, but it will be displayed if XVMAX is less than or equal to XMAX. This virtual maximum is used to determine "nice" numbers for labeling the large tic marks on the x-axis. If XVMAX is not equal to XMAX then the x-axis will not end on a large, labeled, tic mark.

The value of XVMAX is changed to the value of XMAX when the value of XMAX is changed. So, if you want to make XVMAX different from XMAX, it must be changed *after* XMAX is changed.

If NLXINC is set to zero (0), then the x-axis will begin at XMIN and end at XMAX, but if these are not ‘nice’ numbers, the virtual minimum and maximum will be set to values outside the actual minimum and maximum so that the large tic marks can be labeled with "nice" numbers.

XMIN

Default value: XMIN $= 0$

XMIN controls the minimum value for the x-axis.

If $|\mathtt{XLOG}| > 1$, then XMIN is assumed to be the exponent for the minimum value on the x-axis. The minimum value displayed on the x-axis is $|\mathtt{XLOG}|^{\mathtt{XMIN}}$. Integer exponents are always be used for the axis numbering, so the x-axis may not begin on a large tic mark.

If $|\texttt{XLOG}| \leq 1$, `XMIN` is the actual minimum value for the x-axis. If `XVMIN` is equal to `XMIN` then the x-axis always begins on a large, labeled, tic mark. If `XVMIN` is not equal to `XMIN` then the x-axis will not begin at a large, labeled, tic mark.

When the value of `XMIN` is changed, the value of `XVMIN` is simultaneously changed to the same value.

XVMIN

Default value: $\texttt{XVMIN} = 0$

`XVMIN` controls the virtual minimum value for the x-axis. See Figure 14.41 on page 149.

If $|\texttt{XLOG}| > 1$, the virtual minimum is ignored.

If $|\texttt{XLOG}| \leq 1$, then `XVMIN` is the virtual minimum value of the labels for the x-axis. This value will not be displayed if `XVMIN` is less than `XMIN`, but it will be displayed if `XVMIN` is greater than or equal to `XMIN`. This virtual minimum is used to determine "nice" numbers for labeling the large tic marks on the x-axis. If `XVMIN` is not equal to `XMIN` then the x-axis will not begin on a large, labeled, tic mark.

The value of `XVMIN` is changed to the value of `XMIN` when the value of `XMIN` is changed. So, if you want to make `XVMIN` different from `XMIN`, it must be changed *after* `XMIN` is changed.

If `NLXINC` is set to zero, then the x-axis will begin at `XMIN` and end at `XMAX`, but if these are not "nice" numbers, the virtual minimum and maximum will be set to values outside the actual minimum and maximum so that the large tic marks can be labeled with "nice" numbers.

XMOD

Default value: $\texttt{XMOD} = 0$

`XMOD` is another control on the numbering for the x-axis.

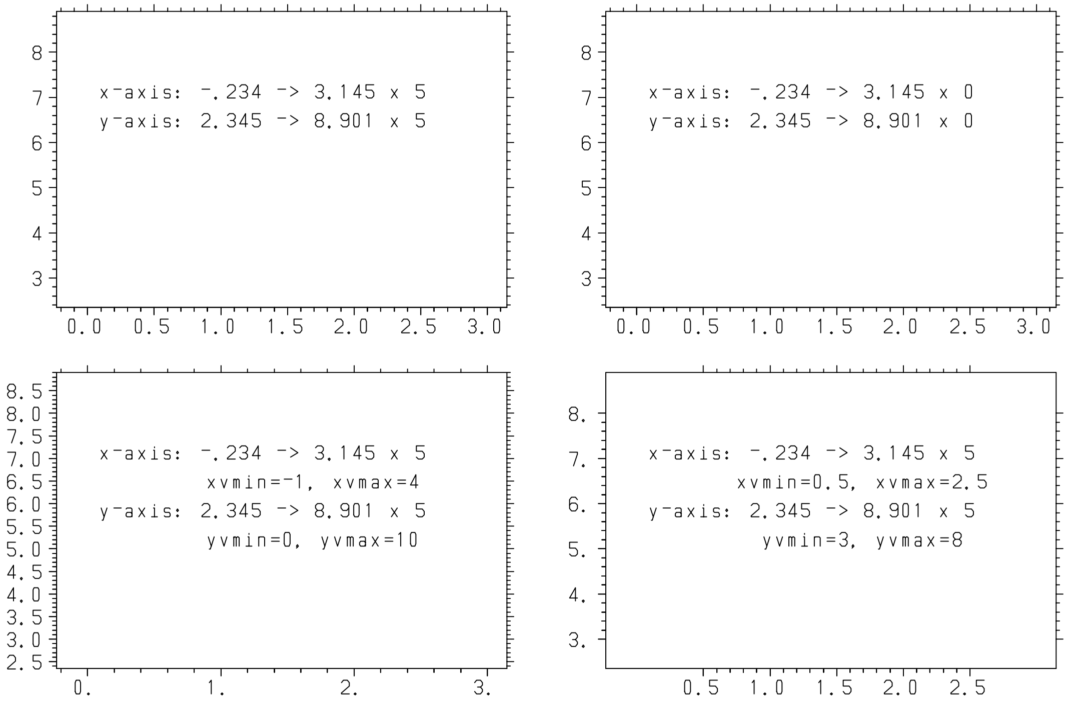


Figure 14.41: Virtual axes examples

$\texttt{XMOD} > 1$	the numbers displayed on the x-axis are modulo `XMOD` and positive. If the number to be displayed on the axis is x then the number that is actually displayed is $\lvert x - \texttt{XMOD} \times (x/\texttt{XMOD})\rvert + \texttt{XOFF}$
$-1 \leq \texttt{XMOD} \leq 1$	the numbers will be displayed normally
$\texttt{XMOD} < -1$	the numbers will be modulo $\lvert\texttt{XMOD}\rvert$ and may be negative

XOFF

Default value: $\texttt{XOFF} = 0$

`XOFF` is another control on the numbering for the x-axis. If $\lvert\texttt{XMOD}\rvert > 1$, then `XOFF` is added to the numbers to be displayed on the x-axis. If $\lvert\texttt{XMOD}\rvert \leq 1$, then `XOFF` is ignored.

XLEADZ

Default value: $\texttt{XLEADZ} = 0$

`XLEADZ` controls whether leading zeros are displayed on the x-axis numbers.

$\texttt{XLEADZ} = 1$	numbers will have leading zeros if they are between 0 and 1
$\texttt{XLEADZ} = 0$	numbers will *not* have leading zeros if they are between 0 and 1

The default is to not display these leading zeros. Numbers that are between -1 and 0 always have the leading zero displayed.

XPAUTO

Default vaule: $\texttt{XPAUTO} = 1$

`XPAUTO`, along with `XPOW`, controls the x-axis scale factor that is appended to the x-axis text label, set by the `XLABEL` keyword using the `SETLAB` routine. This scale factor is needed when there are not enough digits allowed for the numbers labeling the x-axis. `XPOW` is only appended to the text label if $\texttt{XPOW} \neq 0$.

`XPAUTO` $= 2$	determine `XPOW`, but do *not* append the scale factor to the text label
`XPAUTO` $= 1$	determine `XPOW`, and append the scale factor to the text label
`XPAUTO` $= 0$	use the present value of `XPOW`

If the user wishes to completely specify the appearance of the x-axis, `XPAUTO` must be set to 0, otherwise the number of digits and decimal places, `NXDIG` and `NXDEC`, may be changed.

XPOW

Default value: `XPOW` $= 0$

`XPOW` controls the x-axis scale factor that is appended to the x-axis text label, set by the `XLABEL` keyword using the `SETLAB` routine. This scale factor is a power of ten, that is, $10^{\texttt{XPOW}}$, and the numbers labeling the x-axis should be multiplied by this scale factor to get the correct graph units. `XPOW` is only appended to the text label if `XPOW` $\neq 0$.

NXDIG

Default value: `NXDIG` $= 5$

`NXDIG` controls the total number of digits to be displayed in each of the numbers labeling the x-axis. If `NXDIG` is smaller than required to display the x-axis numbers, `NXDIG` will *not* be increased, but a scale factor, $(\times 10^n)$, will be appended to the x-axis text label, set by the `XLABEL` keyword using the `SETLAB` routine. If `NXDIG` is larger than required, `NXDIG` will be reduced to the minimum value required. The value of `NXDIG` is updated after each call to the `GPLOT` routine.

NXDEC

Default value: `NXDEC` $= -1$

`NXDEC` controls the number of digits to be displayed in the fractional parts of the numbers labeling the x-axis. The value of `NXDEC` is updated after each call to the `GPLOT` routine.

NXDEC $= -1$	display the numbers labeling the x-axis as integers, with no decimal point
NXDEC $= 0$	display the numbers labeling the x-axis with no fractional part, but with a decimal point
NXDEC $=> 0$	display the numbers labeling the x-axis with NXDEC digits in the fractional part

XNUMSZ

Default value: %XNUMSZ $= 2$

XNUMSZ controls the size of the numbers labeling the x-axis. See Figure 14.39 on page 139. %XNUMSZ is a percentage of the height of the **GPLOT** window, that is, XNUMSZ $=$ %XNUMSZ $\times$ (YUWIND $-$ YLWIND) $\div 100$

XNUMA

Default value: XNUMA $= -$XAXISA

XNUMA controls the angle of the base line for the numbers labeling the x-axis. XNUMA is the angle, in degrees, measured counterclockwise, between a line parallel to the x-axis and the base line of a number. Refer to Figure 14.39 on page 139. By default, XNUMA is set as a percentage. If XNUMA is set as a percentage, then XNUMA is set to $-$XAXISA. The actual value of %XNUMA will be ignored. This allows the user to change the angle of the x-axis and keep the base line of the x-axis numbers horizontal.

XITICA

Default value: XITICA $=$ YAXISA $-$ XAXISA $+ 180$

XITICA, along with XITICL, controls the location of the numbers labeling the x-axis at the large tic marks. XITICA is the angle, in degrees, measured counterclockwise, between the x-axis and a line joining the base of each large tic mark on the x-axis to the centre of the number labeling that tic mark. See Figure 14.39 on page 139. By default, XITICA is set as a percentage. If XITICA is set as a percentage, then XITICA is set to $\mathtt{YAXISA} - \mathtt{XAXISA} + 180$. The actual value of %XITICA will be ignored. This allows the user to change the angle of the axes and keep the relative locations of the x-axis numbers the same.

XITICL

Default value: $\mathtt{\%XITICL} = 3$

XITICL, along with XITICA, controls the location of the numbers labeling the x-axis at the large tic marks. XITICL is the distance from the base of each large tic mark on the x-axis, to the centre of the number labeling that tic mark. See Figure 14.39 on page 139. %XITICL is a percentage of the height of the **GPLOT** window, that is, $\mathtt{XITICL} = \mathtt{\%XITICL} \times (\mathtt{YUWIND} - \mathtt{YLWIND}) \div 100$

14.7 y-axis keywords

Figure 14.42 on page 154 illustrates the definitions of some of the y-axis keywords.

YAXIS

Default value: $\mathtt{YAXIS} = 1$

YAXIS controls whether or not the y-axis is drawn.

$\mathtt{YAXIS} \neq 0$	draw the y-axis
$\mathtt{YAXIS} = 0$	do not draw the y-axis

If $\mathtt{BOX} \neq 0$, then the left side of the axis box will be drawn, even if $\mathtt{YAXIS} = 0$.

YLABSZ

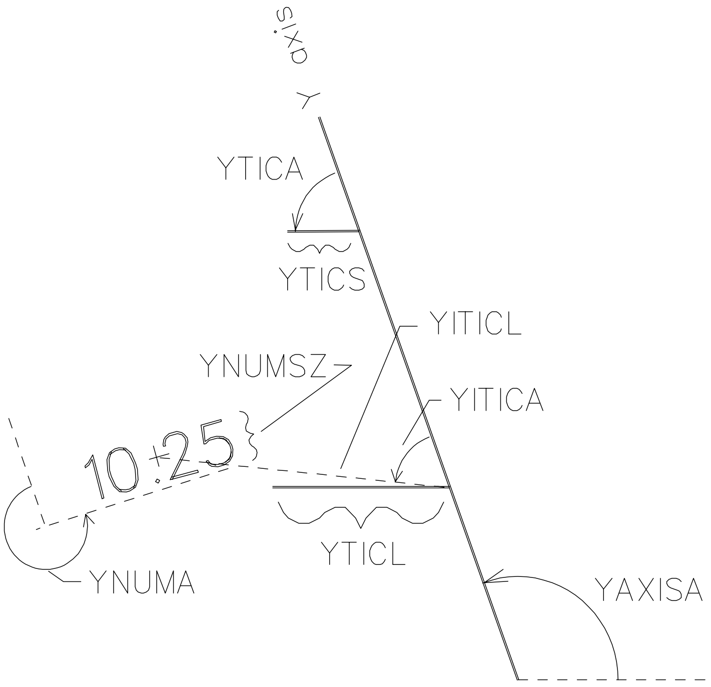


Figure 14.42: Examples of some y-axis characteristics

Default value: `%YLABSZ` $= 3$

`YLABSZ` controls the size of the automatic text label for the y-axis, set by the `YLABEL` keyword using the `SETLAB` routine. `%YLABSZ` is a percentage of the height of the **GPLOT** window, that is, `YLABSZ` $=$ `%YLABSZ` $\times$ (`YUWIND` $-$ `YLWIND`) $\div 100$

YLOG

Default value: `YLOG` $= 0$

`YLOG` determines whether the y-axis is to be linear or logarithmic. See Figure 14.43 on page 156 for examples of various logarithmic axes.

`YLOG` > 1	the y-axis will have a logarithmic scale. The numbers labeling the y-axis will be in exponent form, for example, 10^1 10^2 10^3 10^4.
$-1 \leq$ `YLOG` ≤ 1	the y-axis will have a linear scale
`YLOG` < -1	the y-axis will have a logarithmic scale. The numbers labeling the y-axis will be in full digit form, for example, 10 100 1000 10000.

Suppose that $|\texttt{YLOG}| > 1$. The base will be the *integer part* of $|\texttt{YLOG}|$, except for the special case: $1.05 \times e > \texttt{YLOG} > 0.95 \times e$, where e is the base of the natural logarithms, $e \approx 2.718281828$, in which case the base will be e. `YMIN` and `YMAX` will be the *exponents* for the minimum and maximum values on the y-axis. The maximum value displayed on the y-axis is $|\texttt{YLOG}|^{\texttt{YMAX}}$ and the minimum value displayed on the y-axis is $|\texttt{YLOG}|^{\texttt{YMIN}}$. Integer exponents are always used for the axis numbering, so the y-axis may not begin and/or end on a large tic mark.

If $|\texttt{YLOG}| = 10$, the small tic marks on the y-axis can be specified exactly with `NSYINC`. If small tic marks are desired at the locations $j_m \times 10^n$, where $2 \leq j_m \leq 9$, then set $\texttt{NSYINC} = -j_1 \ldots j_m$. For example, if you want small tic marks at 2×10^n, 4×10^n, 5×10^n, and 8×10^n, then set $\texttt{NSYINC} = -2458$.

NYGRID

Default value: $\texttt{NYGRID} = 0$

`NYGRID` controls the number of grid lines parallel to the x-axis. `NYGRID` specifies the number of large y-axis tic marks between each grid line.

`NYGRID` > 0	draw a grid line parallel to the x-axis at every `NYGRID` large tic mark, starting with the first
`NYGRID` $= 0$	suppress all grid lines parallel to the x-axis
`NYGRID` $= -1$	draw a grid line parallel to the x-axis at every y-axis tic mark, large and small. This option applies only if the axes are orthogonal.

YCROSS

Default value: $\texttt{YCROSS} = 0$

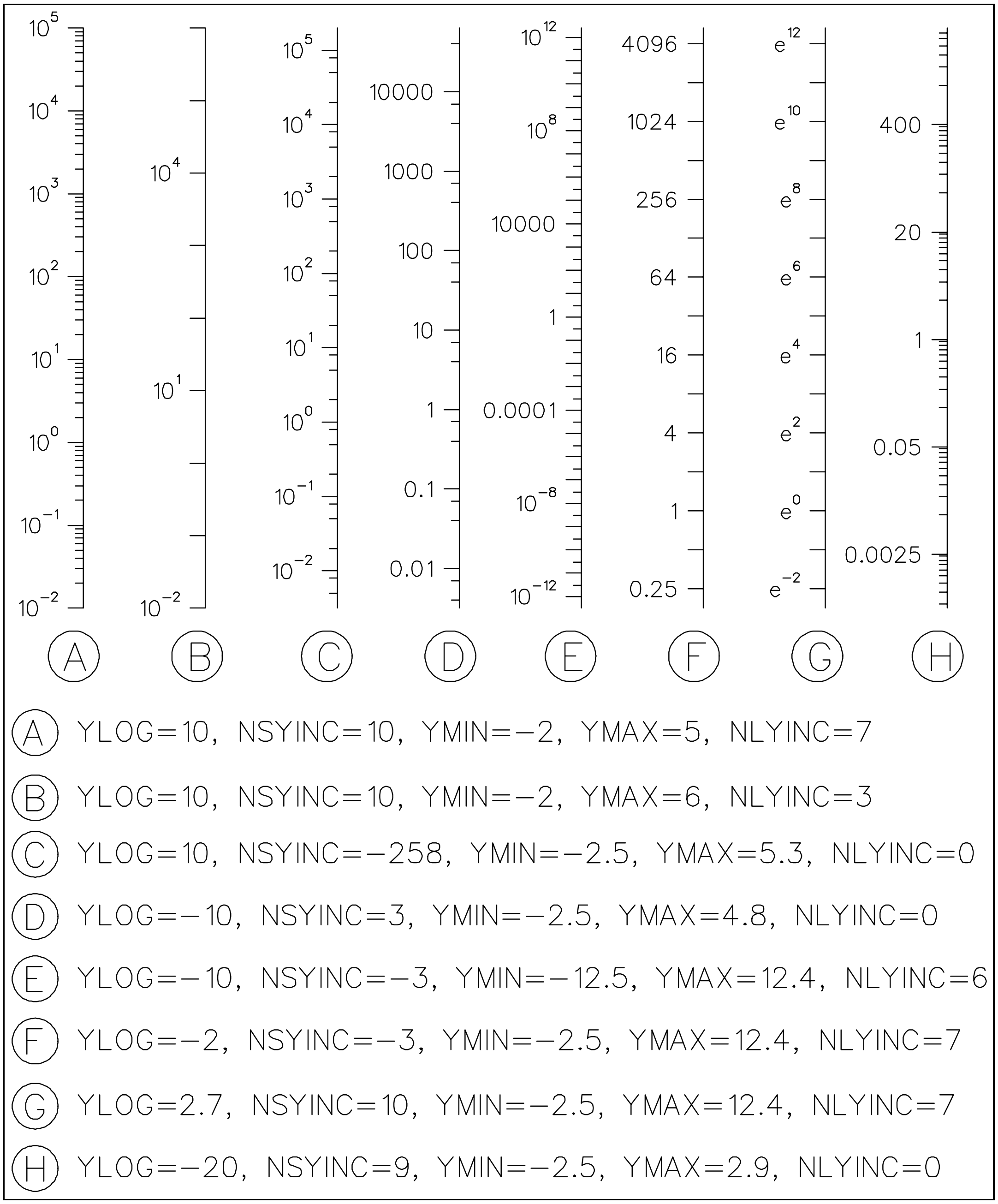

10^5
10^4
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^4
10^1
10^{-2}
10^5
10^4
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10000
1000
100
10
1
0.1
0.01
10^{12}
10^8
10000
1
0.0001
10^{-8}
10^{-12}
4096
1024
256
64
16
4
1
0.25
e^{12}
e^{10}
e^8
e^6
e^4
e^2
e^0
e^{-2}
400
20
1
0.05
0.0025
Ⓐ Ⓑ Ⓒ Ⓓ Ⓔ Ⓕ Ⓖ Ⓗ
Ⓐ YLOG=10, NSYINC=10, YMIN=−2, YMAX=5, NLYINC=7
Ⓑ YLOG=10, NSYINC=10, YMIN=−2, YMAX=6, NLYINC=3
Ⓒ YLOG=10, NSYINC=−258, YMIN=−2.5, YMAX=5.3, NLYINC=0
Ⓓ YLOG=−10, NSYINC=3, YMIN=−2.5, YMAX=4.8, NLYINC=0
Ⓔ YLOG=−10, NSYINC=−3, YMIN=−12.5, YMAX=12.4, NLYINC=6
Ⓕ YLOG=−2, NSYINC=−3, YMIN=−2.5, YMAX=12.4, NLYINC=7
Ⓖ YLOG=2.7, NSYINC=10, YMIN=−2.5, YMAX=12.4, NLYINC=7
Ⓗ YLOG=−20, NSYINC=9, YMIN=−2.5, YMAX=2.9, NLYINC=0

Figure 14.43: Logarithmic y-axis examples

YCROSS controls where the x-axis is to cross the y-axis. The axes always cross at a large tic mark.

YCROSS $= 0$	the x-axis will cross the y-axis at YMIN
YCROSS $\neq 0$	the x-axis will cross the y-axis at $y = 0$, or at the large tic mark closest to $y = 0$

YZERO

Default value: YZERO $= 0$

YZERO controls whether or not to force zero to be displayed on the y-axis. YZERO is set to zero after each call to the GPLOT routine.

YZERO $\neq 0$	if YMIN ≥ 0 then set YMIN to zero, or, if YMAX ≤ 0 then set YMAX to zero
YZERO $= 0$	do not force zero to be displayed on the y-axis

YTICTP

Default value: YTICTP $= 1$

YTICTP controls the type of tic marks to place on the y-axis.

YTICTP $\neq 2$	tic marks are drawn on only one side of the y-axis. YTICA controls the side on which the tic marks will appear
YTICTP $= 2$	tic marks are drawn on both sides of the y-axis

YTICA

Default value: YTICA $=$ XAXISA $-$ YAXISA $+ 180$

`YTICA` controls the angle of the tic marks, both large and small, on the y-axis. `YTICA` is the angle, in degrees, measured counterclockwise, between the y-axis and a tic mark. See Figure 14.42 on page 154. By default, `YTICA` is set as a percentage. If `YTICA` is set as a percentage, then `YTICA` is set to $\texttt{XAXISA} - \texttt{YAXISA} + 180$. The actual value of `%YTICA` will be ignored. This allows the user to change the angle of the axes and keep the y-axis tic marks parallel to the x-axis.

NLYINC

Default value: $\texttt{NLYINC} = 5$

`NLYINC` controls the number of large tic marks to be displayed on the y-axis.

$\texttt{NLYINC} = -1000$	drop the first and the last numbers on the y-axis and determine the number of large y-axis tic marks
$\texttt{NLYINC} < 0$	drop the first and the last numbers on the y-axis. The number of large tic marks on the y-axis will be $\lvert\texttt{NLYINC}\rvert+1$, unless $\texttt{YLOG} > 1$, in which case the number of tic marks will be determined to avoid fractional powers
$\texttt{NLYINC} = 0$	the number of large tic marks on the y-axis will be automatically determined
$\texttt{NLYINC} > 0$	the number of large tic marks on the y-axis will $\texttt{NLYINC}+1$, unless $\texttt{YLOG} > 1$, in which case `NLYINC` will be determined to avoid fractional powers

YTICL

Default value: $\texttt{\%YTICL} = 2$

`YTICL` is the length of the long tic marks on the y-axis. See Figure 14.42 on page 154. `%YTICL` is a percentage of the height of the **GPLOT** window, that is, $\texttt{YTICL} = \texttt{\%YTICL} \times (\texttt{YUWIND} - \texttt{YLWIND}) \div 100$

NSYINC

Default value: `NSYINC` $= 5$

`NSYINC` controls the number of small tic marks to be displayed on the y-axis. The small tic marks appear between the large tic marks. See Figure 14.42 on page 154.

`NSYINC` ≤ 1	no small tic marks on the y-axis
`NSYINC` ≥ 2	the number of small tic marks, on the y-axis, between each pair of large tic marks, will be `NSYINC` -1

If $|$`YLOG`$| = 10$, the small tic marks on the y-axis can be specified exactly with `NSYINC`. If small tic marks are desired at the locations $j_m \times 10^n$, where $2 \leq j_m \leq 9$, then set `NSYINC` $= -j_1 \ldots j_m$. For example, if you want small tic marks at 2×10^n, 4×10^n, 5×10^n, and 8×10^n, then set `NSYINC` $= -2458$.

YTICS

Default value: `%YTICS` $= 1$

`YTICS` is the length of the short tic marks on the y-axis. See Figure 14.42 on page 154. `%YTICS` is a percentage of the height of the **GPLOT** window, that is, `YTICS` $=$ `%YTICS` $\times$ (`YUWIND` $-$ `YLWIND`) $\div 100$

YAUTO

Default value: `YAUTO` $= 2$

`YAUTO` controls the autoscaling of the y-axis. `YAUTO` is ignored if the `GPLOT` argument $iaxis = 2$, that is, if only the data points are to be plotted and not the axes.

`YAUTO` is set to zero after each call to `GPLOT`.

`YAUTO` $= 3$	set `YMIN` and `YMAX` to the minimum and maximum of the data passed to the `GPLOT` routine in the y array. Determine `YVMIN`, `YVMAX`, and `NLYINC` so that ‘nice’ numbers will label the large tic marks on the y-axis. The y-axis might not begin and end on large, labeled, tic marks. Also determine `YPOW`, `NYDIG` and `NYDEC`. Do not set `YPAUTO` to 1 or 2 after setting `YAUTO` to 3
`YAUTO` $= 2$	determine `YMIN` and `YMAX` so that all the data points passed to `GPLOT` in the y array appears on the graph. Set `YVMIN` to `YMIN` and set `YVMAX` to `YMAX` so that the y-axis will begin and end on large, labeled, tic marks. Determine `NLYINC` so that ‘nice’ numbers will label the large tic marks on the y-axis. Also determine `YPOW`, `NYDIG` and `NYDEC`. Do not set `YPAUTO` to 1 or 2 after setting `YAUTO` to 2.
`YAUTO` $= 1$	determine `YMIN` and `YMAX`. Set `YVMIN` to `YMIN` and set `YVMAX` to `YMAX` so the y-axis will begin and end at large, labeled, tic marks. If `NLYINC` $= 0$ then determine `NLYINC` so that ‘nice’ numbers will label the large tic marks on the y-axis. Use the current values for `YPOW`, `NYDIG` and `NYDEC`.
`YAUTO` $= 0$	do not autoscale the y-axis, that is, use the current values for `YMIN` and `YMAX`. If `NLYINC` $= 0$, determine `YVMIN`, `YVMAX`, and `NLYINC` so that ‘nice’ numbers will label the large tic marks on the y-axis. The y-axis may not begin and end on large, labeled, tic marks.

If `YMIN` $=$ `YMAX`, then `YMIN` and `YMAX` will be determined from the x and y arrays passed to the `GPLOT` routine, just as if `YAUTO` had been set to one (1).

YMAX

Default value: `YMAX` $= 10$

`YMAX` is the maximum value on the y-axis.

If $|\texttt{YLOG}| > 1$, then `YMAX` is assumed to be the *exponent* for the maximum value on the y-axis. The maximum value displayed on the y-axis is $|\texttt{YLOG}|^{\texttt{YMAX}}$. Integer exponents are always used for the axis numbering, so the y-axis may not end on a large tic mark.

If $|\texttt{YLOG}| \le 1$, then `YMAX` is the actual maximum value for the y-axis. If `YVMAX` is equal to `YMAX` then the y-axis always ends on a large, labeled, tic mark. If `YVMAX` is not equal to `YMAX` then the y-axis will not end at a large, labeled, tic mark.

When the value of `YMAX` is changed, the value of `YVMAX` is simultaneously changed to the same value.

YVMAX

Default value: `YVMAX` $= 10$

`YVMAX` is the virtual maximum value for the y-axis. See Figure 14.41 on page 149.

If $|$`YLOG`$| > 1$, the virtual maximum is ignored.

If $|$`YLOG`$| \leq 1$, then `YVMAX` is the virtual maximum value of the labels for the y-axis. This value will not be displayed if `YVMAX` is greater than `YMAX`, but it will be displayed if `YVMAX` is less than or equal to `YMAX`. This virtual maximum is used to determine "nice" numbers for labeling the large tic marks on the y-axis. If `YVMAX` is not equal to `YMAX` then the y-axis will not end on a large, labeled, tic mark.

The value of `YVMAX` is changed to the value of `YMAX` when the value of `YMAX` is changed. So, if you want to make `YVMAX` different from `YMAX`, it must be changed *after* `YMAX` is changed.

If `NLYINC` is set to zero, then the y-axis will begin at `YMIN` and end at `YMAX`, but if these are not "nice" numbers, the virtual minimum and maximum will be set to values outside the actual minimum and maximum so that the large tic marks can be labeled with "nice" numbers.

YMIN

Default value: `YMIN` $= 0$

`YMIN` is the minimum value on the y-axis.

If $|$`YLOG`$| > 1$, then `YMIN` is assumed to be the *exponent* for the minimum value on the y-axis. The minimum value displayed on the y-axis is $|\mathtt{YLOG}|^{\mathtt{YMIN}}$. Integer exponents are always be used for the axis numbering, so the y-axis may not begin on a large tic mark.

If $|$`YLOG`$| \leq 1$, `YMIN` is the actual minimum value for the y-axis. If `YVMIN` is equal to `YMIN` then the y-axis always begins on a large, labeled, tic mark. If `YVMIN` is not equal to `YMIN` then the y-axis will not begin at a large, labeled, tic mark.

When the value of `YMIN` is changed, the value of `YVMIN` is simultaneously changed to the same value.

YVMIN

Default value: `YVMIN` $= 0$

`YVMIN` is the virtual minimum value on the y-axis. See Figure 14.41 on page 149.

If $|\texttt{YLOG}| > 1$, the virtual minimum is ignored.

If $|\texttt{YLOG}| \leq 1$, then `YVMIN` is the virtual minimum value of the labels for the y-axis. This value will not be displayed if `YVMIN` is less than `YMIN`, but it will be displayed if `YVMIN` is greater than or equal to `YMIN`. This virtual minimum is used to determine "nice" numbers for labeling the large tic marks on the y-axis. If `YVMIN` is not equal to `YMIN` then the y-axis will not begin on a large, labeled, tic mark.

The value of `YVMIN` is changed to the value of `YMIN` when the value of `YMIN` is changed. So, if you want to make `YVMIN` different from `YMIN`, it must be changed *after* `YMIN` is changed.

If `NLYINC` is set to zero, then the y-axis will begin at `YMIN` and end at `YMAX`, but if these are not "nice" numbers, the virtual minimum and maximum will be set to values outside the actual minimum and maximum so that the large tic marks can be labeled with "nice" numbers.

YMOD

Default value: `YMOD` $= 0$

`YMOD` is another control on the numbering for the y-axis.

`YMOD` > 1	the numbers displayed on the y-axis will be modulo `YMOD` and positive. If the number to be displayed on the axis is y then the number that is actually displayed is $|y - \texttt{YMOD} \times (y/\texttt{YMOD})| + \texttt{YOFF}$
$-1 \leq$ `YMOD` ≤ 1	the numbers will be displayed normally
`YMOD` < -1	the numbers will be modulo $|\texttt{YMOD}|$ and may be negative

YOFF

Default value: `YOFF` $= 0$

`YOFF` is another control on the numbering for the y-axis. If $|$`YMOD`$| > 1$, then `YOFF` is added to the numbers to be displayed on the y-axis. If $|$`YMOD`$| \leq 1$, then `YOFF` is ignored.

YLEADZ

Default value: `YLEADZ` $= 0$

`YLEADZ` controls whether leading zeros are displayed on the numbers labeling the y-axis.

`YLEADZ` $\neq 0$	the numbers displayed on the will have leading zeros if they are between 0 and 1
`YLEADZ` $= 0$	the numbers displayed on the will *not* have leading zeros if they are between 0 and 1

The default is to not display these leading zeros. Numbers that are between -1 and 0 always have the leading zero displayed.

YPAUTO

Default vaule: `YPAUTO` $= 1$

`YPAUTO`, along with `YPOW`, controls the y-axis scale factor that is appended to the y-axis text label, set by the `YLABEL` keyword using the `SETLAB` routine. This scale factor is needed when there are not enough digits allowed for the numbers labeling the y-axis.

`YPAUTO` $= 2$	determine `YPOW`, but do not append the scale factor to the text label even if it is needed
`YPAUTO` $= 1$	determine `YPOW`, and append the scale factor to the text label
`YPAUTO` $= 0$	use the present value of `YPOW` and if `YPOW` $\neq 0$ append the scale factor to the text label

If the user wishes to completely specify the appearance of the y-axis, `YPAUTO` must be set to 0, otherwise the number of digits and decimal places, `NYDIG` and `NYDEC`, may be changed.

YPOW

Default value: `YPOW` $= 0$

`YPOW` controls the y-axis scale factor that is appended to the y-axis text label, set by the `YLABEL` keyword using the `SETLAB` routine. This scale factor is a power of ten, that is, $10^{\mathtt{YPOW}}$, and the numbers labeling the y-axis should be multiplied by this scale factor to get the correct graph units.

NYDIG

Default value: `NYDIG` $= 5$

`NYDIG` controls the total number of digits to be displayed in each of the numbers labeling the y-axis. If `NYDIG` is smaller than required to display the y-axis numbers, `NYDIG` will *not* be increased but a scale factor, $(\times 10^n)$, will be appended to the y-axis text label, set by the `YLABEL` keyword using the `SETLAB` routine. If `NYDIG` is larger than required, `NYDIG` will be reduced to the minimum value required. The value of `NYDIG` is updated after each call to the `GPLOT` routine.

NYDEC

Default value: `NYDEC` $= -1$

`NYDEC` controls the number of digits to be displayed in the fractional parts of the numbers labeling the y-axis. The value of `NYDEC` is updated after each call to the `GPLOT` routine.

$\texttt{NYDEC} = -1$	display the numbers labeling the y-axis as integer with no decimal point
$\texttt{NYDEC} = 0$	display the numbers labeling the y-axis with no fractional part, but with a decimal point
$\texttt{NYDEC} > 0$	display the numbers labeling the y-axis with `NYDEC` digits in the fractional part

YNUMSZ

Default value: $\texttt{\%YNUMSZ} = 2$

`YNUMSZ` controls the size of the numbers labeling the y-axis. See Figure 14.42 on page 154. `%YNUMSZ` is a percentage of the height of the **GPLOT** window, that is, $\texttt{YNUMSZ} = \texttt{\%YNUMSZ} \times (\texttt{YUWIND} - \texttt{YLWIND}) \div 100$

YNUMA

Default value: $\texttt{YNUMA} = -\texttt{YAXISA}$

`YNUMA` controls the angle of the base line for the numbers labeling the y-axis. `YNUMA` is the angle, in degrees, measured counterclockwise, between a line parallel to the y-axis and the base line of a number. See Figure 14.42 on page 154. By default, `YNUMA` is set as a percentage. If `YNUMA` is set as a percentage, then `YNUMA` is set to $-\texttt{YAXISA}$. The actual value of `%YNUMA` will be ignored. This allows the user to change the angle of the y-axis and keep the base line of the y-axis numbers horizontal.

YITICA

Default value: $\texttt{YITICA} = \texttt{XAXISA} - \texttt{YAXISA} + 180$

`YITICA`, along with `YITICL`, controls the location of the numbers labeling the y-axis at the large tic marks. `YITICA` is the angle, in degrees, measured counterclockwise, between the y-axis and a line joining the base of each large tic mark on the y-axis to the center of the number labeling that tic mark. See Figure 14.42 on page 154. By default, `YITICA` is set as a percentage. If `YITICA` is set as a percentage, then `YITICA` is set to $\texttt{XAXISA} - \texttt{YAXISA} + 180$. The actual value of `%YITICA` will be ignored. This allows the user to change the angle of the axes and keep the relative location of the y-axis numbers the same.

YITICL

Default value: $\texttt{\%YITICL} = 3$

`YITICL`, along with `YITICA`, controls the location of the numbers labeling the y-axis at the large tic marks. `YITICL` is the distance from the base of each large tic mark on the y-axis to the lower left hand corner of the number labeling that tic mark. See Figure 14.42 on page 154. `%YITICL` is a percentage of the height of the **GPLOT** window, that is, $\texttt{YITICL} = \texttt{\%YITICL} \times (\texttt{YUWIND} - \texttt{YLWIND}) \div 100$

15 GPLOT EXAMPLES

VMS

```
$FORTRAN PROG.FOR
$LINK PROG,GPLOT$DIR:GPLOT/LIB
$RUN PROG
```

The logical name `GPLOT$DIR` points to the location of the **GPLOT** library, `GPLOT.OLB`.

UNIX

```
%f77 -static -c prog.f
%f77 -o prog prog.o -lgplot -lX11
%prog
```

The library `gplot.a` must be installed as `/usr/local/lib/libgplot.a/`. Otherwise, an explicit pathname must be given for `gplot.a`

OSF1

```
%f77 -static -c prog.f
%f77 -o prog -T 00400000 -D 10000000 prog.o -lgplot -lX11
%prog
```

The `-T -D` flags force the addresses into the two gigabyte range, which is necessary for 32 bit addressing.

The library `gplot.a` must be installed as `/usr/local/lib/libgplot.a/`
Otherwise, an explicit pathname must be given for `gplot.a`

15.1 A simple GPLOT example

An example of the output from the following program is shown in Figure 15.44 on page 168.

```
      REAL*4 X(100), Y(100)

C  X window monitor, PostScript hardcopy

      CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
      CALL CLEAR_PLOT
      DO I = 1, 100
        X(I) = I
        Y(I) = SIN(I/10.)/SQRT(I/5.)
      END DO

C  Plot 100 points of X versus Y using all the GPLOT defaults
C  Note: plot the axes first and then the data curve

      CALL NARGSI(4)
      CALL GPLOT(X,Y,100,1)
      CALL NARGSI(1)
      CALL GRAPHICS_HARDCOPY(0)
      STOP
      END
```

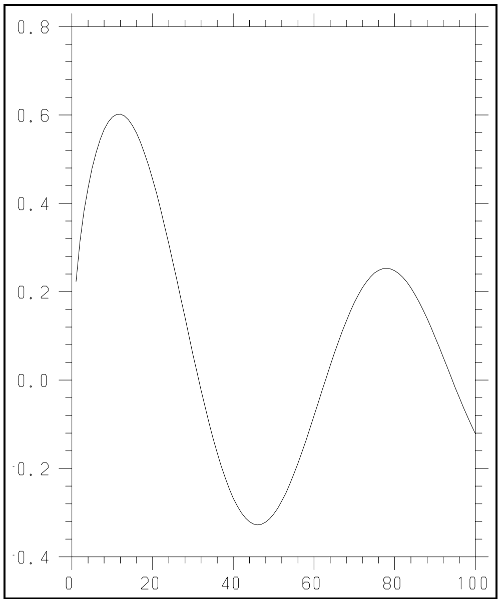


Figure 15.44: A simple GPLOT example

15.2 A GPLOT example with error bars

An example of the output from the following program is shown in Figure 15.45 on page 170.

```
      REAL*4    X(100,3), Y(100,3)
      INTEGER*4 NPT(2)

C  X window monitor, PostScript hardcopy

      CALL SET_PLOT_DEVICES(18,6,14,7,0,'CM','PORTRAIT',1)
      CALL CLEAR_PLOT

C    NPT(1) is the number of points to plot, while NPT(2) is the
C    actual first dimension of arrays X and Y, as declared above

      NPT(1) = 10
      NPT(2) = 100

C    Define the independent variable data in X(I,1) and the dependent
C    variable data in Y(I,1) for I = 1 to 10.
C    Define the lower and upper Y error in Y(I,2) and Y(I,3) and the
C    lower and upper X error in X(I,2) and X(I,3)

      ISEED = 12345678
      XTEMP = RAN(ISEED)
      DO I = 1, 10
        X(I,1) = I
        Y(I,1) = SIN(X(I,1))/SQRT(X(I,1)*2.)
        X(I,2) = RAN(ISEED)
        X(I,3) = RAN(ISEED)
        Y(I,2) = RAN(ISEED)/8.
        Y(I,3) = RAN(ISEED)/8.
      END DO
      CALL SETNAM('ERRBAR',6.)   ! Asymmetric X and Y error bars
      CALL SETLAB('HEXCHR','01') ! Set plotting symbol to a "box"
      CALL SETNAM('PMODE',-1.)   ! Unjoined plotting symbols
      CALL SETNAM('MASK',1.)
      CALL SETNAM('%YUWIND',50.)
      CALL SETLAB('XLABEL','Asymmetric Error Bars')
      CALL SETLAB('FONT','TSAN')
      CALL GPLOT(X,Y,NPT,1)
      CALL SETLAB('HEXCHR','0C')  ! Set plotting symbol to a "circle"
      CALL SETNAM('%YLWIND',50.)
      CALL SETNAM('%YUWIND',100.)
      CALL SETNAM('ERRBAR',3.)    ! Symmetric X and Y error bars
      CALL SETLAB('XLABEL','Symmetric Error Bars')
      CALL NARGSI(4)
      CALL GPLOT(X,Y,NPT,1)
      CALL NARGSI(1)
      CALL GRAPHICS_HARDCOPY(0)
      STOP
      END
```

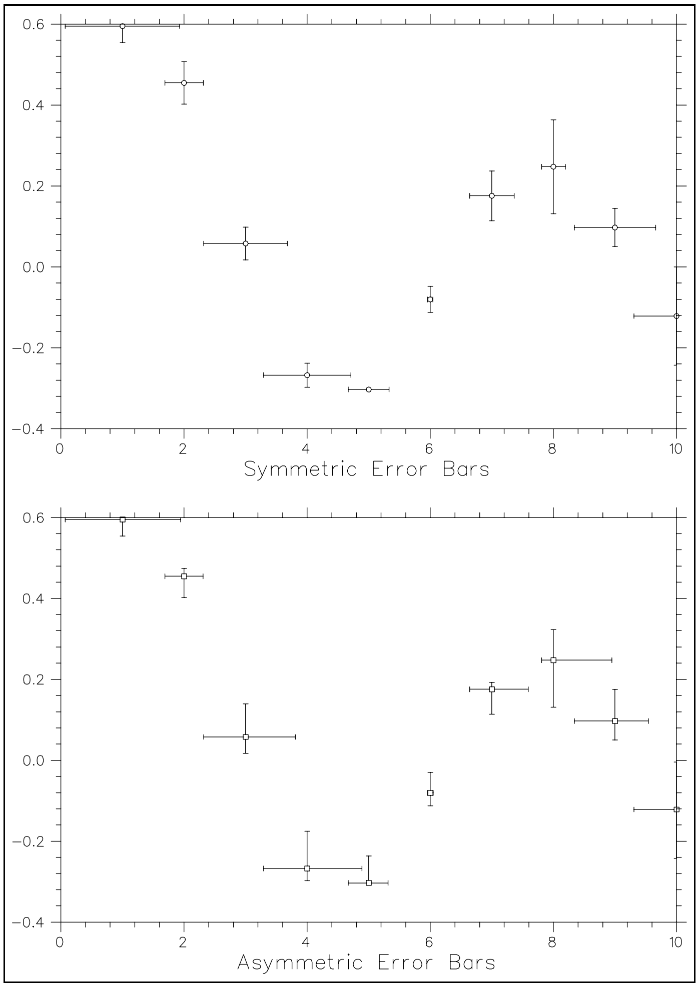


Figure 15.45: A `GPLOT` example with error bars

15.3 A GPLOT example with multiple curves and axes

An example of the output from the following program is shown in Figure 15.46 on page 173.

```
      REAL*4    X(50), Y1(50), Y2(50), Y3(50)
      LOGICAL*1 PLOT_CHAR(50)
      CALL GPLOT_SETUP(' ')
      NPT = 50
      DO I = 1, NPT
C  Set the plot character to a box if I even, to a star if I odd
        IF( MOD(I,2) .EQ. 0 )THEN
          PLOT_CHAR(I) = 1
        ELSE
          PLOT_CHAR(I) = 14
        END IF
C    Artificially create some data
        X(I)  = I*10.
        Y1(I) = ABS(SIN(I/10.))
        Y2(I) = I+20.*SIN(I/5.)
        Y3(I) = I
      END DO
      CALL SETLAB('FONT','TSAN')
      CALL SETNAM('%CHARSZ',2.)  ! Plot character size = 2% of window
      CALL SETNAM('%XTICL',1.5)  ! x-axis major tic length = 1.5% of window
      CALL SETNAM('%YTICL',1.5)  ! Y axis major tic length = 1.5% of window
      CALL SETNAM('XLOG',-10.)   ! Make x-axis logarithmic with base 10
      CALL SETNAM('TOPNUM',-1.)  ! Number outside of top edge of box
      CALL SETNAM('RITNUM',-1.)  ! Number outside of right edge of box
C   Use the bottom half of the screen
      CALL SETNAM('%YLWIND',10.)
      CALL SETNAM('%YUWIND',50.)
      CALL SETNAM('NLXINC',3.)   ! # of long X increments = 3
      CALL SETNAM('NLYINC',5.)   ! # of long Y increments = 5
      CALL SETNAM('NSYINC',5.)   ! # of short Y increments = 5
      CALL SETNAM('MASK',-1.)    ! Use PLOT_CHAR for the plotting char's.
C   Produce the first graph
      CALL NARGSI(5)
      CALL GPLOT(X,Y1,NPT,1,PLOT_CHAR)
      CALL SETNAM('YAUTO',2.)    ! Auto scale Y axis
      CALL SETNAM('YLOG',-2.)    ! Make Y axis logarithmic with base 2
C   Use the top half of the screen
      CALL SETNAM('%YLWIND',45.)
      CALL SETNAM('%YUWIND',100.)
      CALL SETNAM('%XNUMSZ',0.)
      CALL SETNAM('MASK',0.)     ! Use no plotting symbol
C   Produce the second graph
```

```
      CALL NARGSI(4)
      CALL GPLOT(X,Y2,NPT,1)
      CALL SETNAM('PMODE',-1.)   ! Do not connect the data points
      CALL SETLAB('HEXCHR','010A')
      CALL SETNAM('MASK',10.)    ! Plot every 10th point with the triangle
      CALL SETNAM('%CHARSZ',2.)
C   Overlay the third graph on the second
C   No axes to be plotted this time  ... IAXIS = 2
      CALL NARGSI(4)
      CALL GPLOT(X,Y3,NPT,2)
      CALL SETNAM('%XLOC',50.)   ! Set X location of text to 50% of window
      CALL SETNAM('%YLOC',2.)    ! Set Y location of text to 2% of window
      CALL SETNAM('CURSOR',-2.)  ! Use centre justification for text
C   Reset to the full window
      CALL SETNAM('%YLWIND',0.)
      CALL SETNAM('%YUWIND',100.)
C   Plot a text string at the bottom of the window
C    at position XLOC, YLOC with the default height and angle
      CALL SETLAB('TEXT','Example of GPLOT usage')
      CALL NARGSI(1)
      CALL GRAPHICS_HARDCOPY(0)
      STOP
      END
```

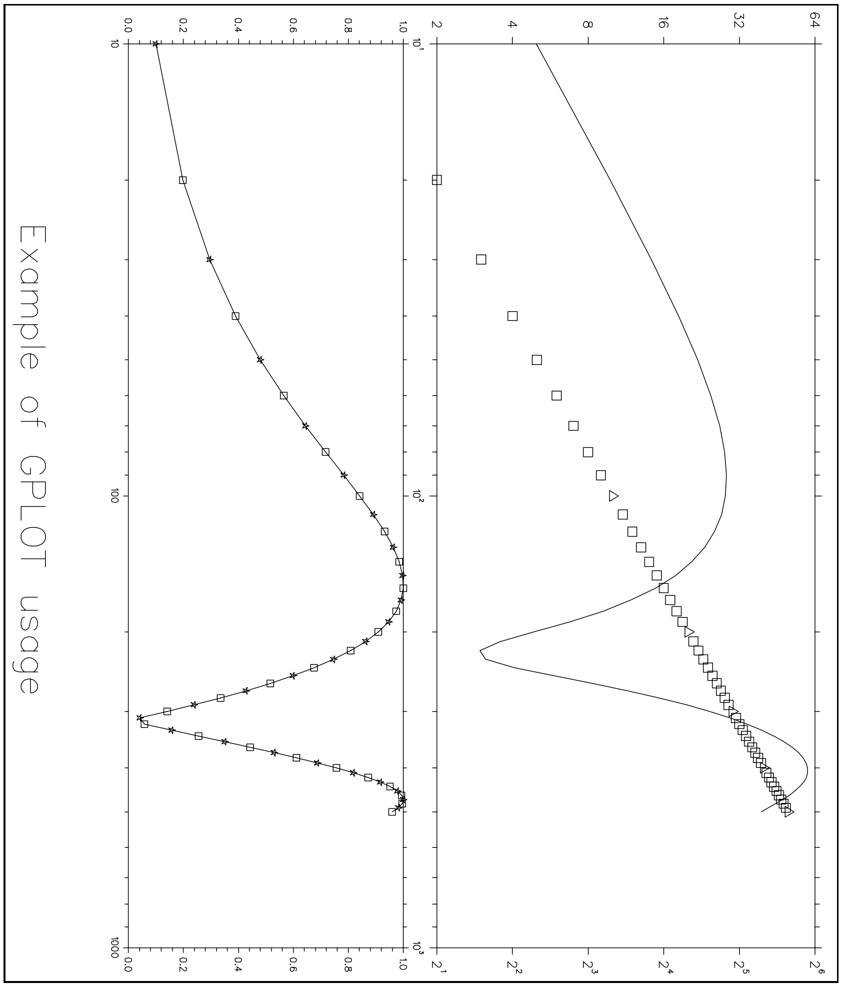


Figure 15.46: A `GPLOT` example with multiple curves and axes

15.4 A GPLOT example with filled histogram bars

An example of the output from the following program is shown in Figure 15.47 on page 175. Note that since an **EDGR** drawing file is opened, no hardcopy facility need be added to the program.

```
      REAL*4    X(50), Y(50)
      LOGICAL*1 PCHAR(50)
C  Set up the device configuration and the world coordinate system 19 x 25 cm
      CALL GPLOT_SETUP(' ')
      SF = 25./640.             ! Scale fill patterns from 480x640 to 19x25
      DO I = 1, 10
        CALL HATCH_SCALE(I,SF)
      END DO
      DO I = 1, 50              !  Artificially create some data
        X(I) = I*10.
        Y(I) = ABS( SIN(I/10.) )
        PCHAR(I) = MOD(I,10)+1
      END DO
      CALL SETLAB('FONT','TSAN')
      CALL SETNAM('HISTYP',2.0)
      CALL SETNAM('MASK',-1.0)
      CALL DWG_BATCH('EXAMPLE4',IER)  !  Open an EDGR file
      CALL NARGSI(5)
      CALL GPLOT(X,Y,50,1,PCHAR)      !  Produce the graph
      STOP
      END
```

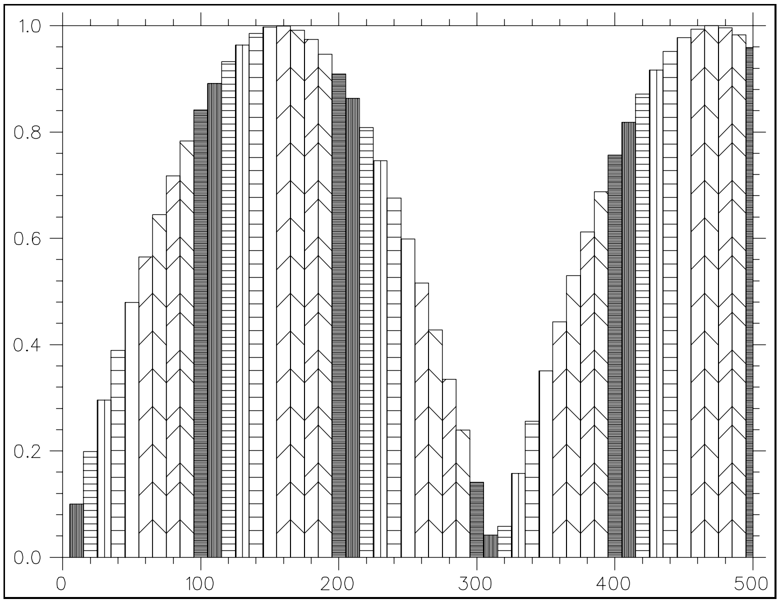


Figure 15.47: A GPLOT example with filled histogram bars

15.5 A GPLOT example of area filling

An example of the output from the following program is shown in Figure 15.48 on page 177. Note that since an **EDGR** drawing file is opened, no hardcopy facility need be added to the program.

```
      REAL*4 X(200), Y(200)
C  Set up the device configuration:
C   X window monitor, no second monitor, no bitmap
C   units are inches, portrait mode
      CALL SET_PLOT_DEVICES(18,6,0,7,12,'IN','PORTRAIT',1)
      CALL CLEAR_PLOT
      SF = 10./640.                   !  Scale fill patterns to 8x10 inch world
      DO I = 1, 10
        CALL HATCH_SCALE(I,SF)
      END DO
      DO I = 1, 100                   !  Artificially create some data
        X(I) = (I-1)*200./99.
        Y(I) = 3.*SIND(X(I))
      END DO
      DO I = 101, 199
        X(I) = X(200-I)
        Y(I) = SIND(X(I)*3./2.)
      END DO
      CALL DWG_BATCH('GPLOT_EX5',IER) !  Open an EDGR file
      CALL DWG_FORMAT('PORTRAIT',1)
      CALL SETLAB('FONT','TRIUMF.2')  !  Set the text font to TRIUMF.2
      CALL SETNAM('LINTYP',103.)      !  Use fill pattern number  103-100 = 3
      CALL SETNAM('%YLAXIS',55.)      !  Use the top half of the page
      CALL NARGSI(4)
      CALL GPLOT(X,Y,199,1)           !  Plot the graph
      DO I = 1, 10                    !  Artificially create some more data
        X(I) = (I-1)*200./9.
        Y(I) = 3.*SIND(X(I))
      END DO
      X(11) = X(10)
      Y(11) = 0.0
      X(12) = 500.
      Y(12) = 0.0
      X(13) = X(1)
      Y(13) = 0.0
C  Use the bottom half of the page
      CALL SETNAM('%YUAXIS',45.)
      CALL SETNAM('%YLAXIS',15.)
      CALL SETNAM('HISTYP',1.)        !  Plot a histogram with no tails
      CALL SETNAM('LINTYP',108.)      !  Use fill pattern number  108-100 = 8
C  Set the graph scales
      CALL SETNAM('XMIN',0.0)
      CALL SETNAM('XMAX',250.)
      CALL SETNAM('NLXINC',5.0)
      CALL SETNAM('YMIN',-2.0)
      CALL SETNAM('YMAX',4.0)
      CALL SETNAM('NLYINC',6.0)
      CALL NARGSI(4)
      CALL GPLOT(X,Y,13,1)            !  Plot the graph
      END
```

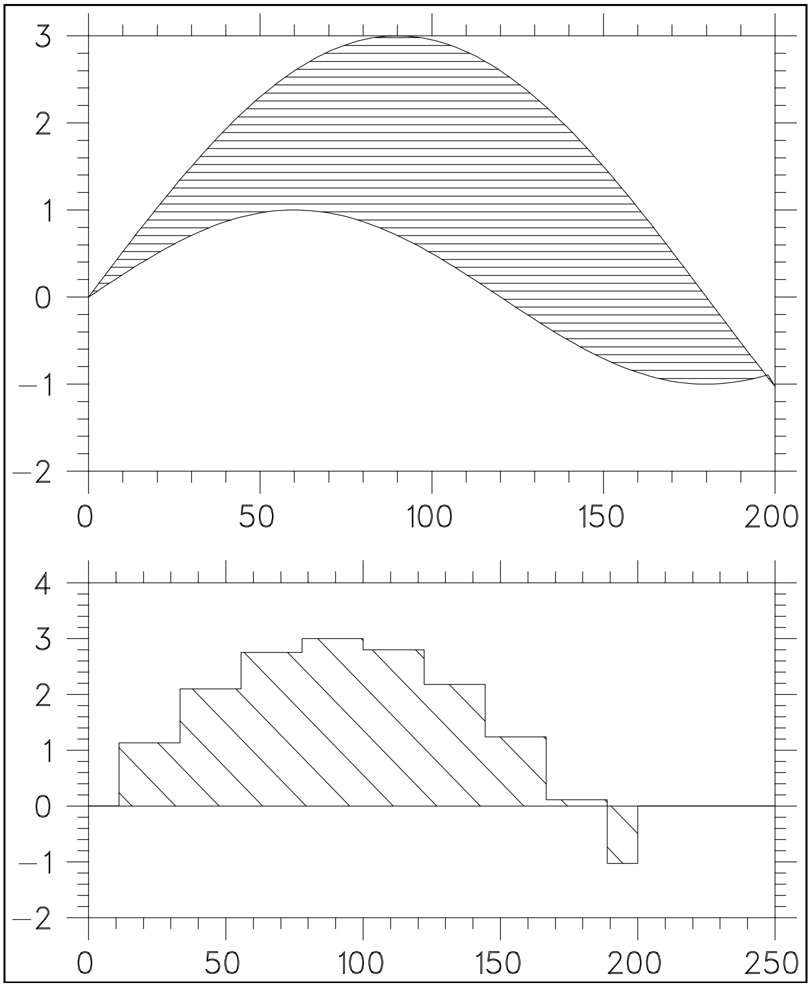


Figure 15.48: A GPLOT example of area filling

A GENERIC TERMINAL FILE DESCRIPTION

The name of the generic terminal file is found by translating a logical name on VMS systems, and by translating an environment variable, on UNIX systems.

VMS
```
$ DEFINE TRIUMF$GENTERM disk:[directory]filename.ext
```

UNIX
```
% setenv TRIUMF_GENTERM \path\filename
```

Example files for VT640, CIT467 and VT241 terminals, and for the KERMIT terminal emulator with VGA colour, are listed in **Appendix B**.

A terminal characteristics file should contain ‘commands’ and ‘parameter(s)’. For example:

`VECTOR_MODE` ⟸ *command*
`<GS>` ⟸ *parameter*

If any of the ‘command’s are not in the file, their parameter(s) will assume default value(s). Defaults for each ‘command’ are listed below.

The order that the ‘commands’ are encountered in the file is irrelevant.

Lines that start with `!` are considered to be comment lines. Comment lines *cannot* be inserted between a ‘command’ and its following ‘parameter’ line(s).

Control sequences can be input in four ways, whatever is most convenient:

1. the literal character
2. the ASCII decimal code
3. the standard name
4. the control equivalence

Choices (2), (3) and (4) *must* be enclosed within $<$ and $>$.

For example, escape has the decimal ASCII code 27 and is the same as control-[. So you can enter it into the file as

1. the literal escape character
2. `<27>`
3. `<ESC>`
4. `<CTRL[>`

The basic type of terminal must be similar to Tektronix 4010, 4014, or Regis. For example: VT640 and CIT467 are Tektronix 4010 type, while VT241 is a Regis type.

Terminal type. The following choices are currently available: `TEK4010`, `TEK4014`, `REGIS`.

Default: `keyword = TEK4010`

```
TERMINAL_TYPE
keyword
```

Size of the graphics screen in pixels.

Default: `xmin = 0.0, xmax = 640.0, ymin = 0.0, ymax = 479.0`

```
RANGE
xmin  xmax  ymin  ymax
```

Clear terminal graphics screen.

Default: `sequence =  <ESC><FF>`

```
GRAPHICS_CLEAR
sequence
```

Clear terminal alphanumeric (transparent) screen.

Default: `sequence =  <CAN><ESC>[2J<ESC>[1;1H`

```
TRANSPARENT_CLEAR
sequence
```

Enter alphanumeric (transparent) mode.

Default: `sequence = <CAN>`

`ENTER_TRANS_MODE`
`sequence`

Turn graphics pixels on.

Default: `sequence = <ESC>/0d`

`DOTS_ON`
`sequence`

Turn graphics pixels off.

Default: `sequence = <ESC>/1d`

`DOTS_OFF`
`sequence`

Complement graphics pixels.

Default: `sequence = <ESC>/2d`

`DOTS_COMPLEMENT`
`sequence`

Loading the graphics crosshair requires two control sequences: the first sequence is followed by the HI y and x and LO y and x and is followed by the second sequence. Conversion factors and offsets are needed for mapping.

```
Defaults: first_sequence  = <GS>
          second_sequence = <ESC>/f<ESC><SUB>
          xfrac_1 = 1.60          = 1024/640
          xoff_1  = 0.0           no offset
          yfrac_1 = 1.625         =  780/480
          yoff_1  = 0.0           no offset
          xfrac_2 = 0.625         =  640/1024
          xoff_2  = 0.6
          yfrac_2 = 0.615384615   =  480/780
          yoff_2  = 0.6
```

```
CROSSHAIRS
first_sequence
second_sequence
xfrac_1 xoff_1
yfrac_1 yoff_1
xfrac_2 xoff_2
yfrac_2 yoff_2
```

Special lines are written immediately to the terminal in character format. This is necessary for initializing a some terminals, for example, CIT467's require `<ESC>1<ESC>O !VEC 0,0,0,0 !BYE`.

There is no default.

```
SPECIAL
line
```

Terminal graphics colour. Enter the initial and final colour number followed by colour sequences. The maximum allowable number of colours is 0 to 20.

Default: `no colours (assumes none available)`

```
COLOURS
init_colour final_colour
sequence_for_colour
sequence_for_colour
sequence_for_colour
...
```

Toggle off graphics without clearing (available on CIT467's).

Default: `none (assumes not available)`

```
VIDEO_OFF
sequence
```

Toggle graphics back on after `VIDEO_OFF` (available on CIT467's).

Default: `none (assumes not available)`

```
VIDEO_ON
sequence
```

Plot monitor horizontal and vertical scale factors and offsets.

```
 Defaults:  xfrac = 1.60          = 1024/640
            xoff  = 0.5           offset
            yfrac = 1.625         =  780/480
            yoff  = 0.5           offset
```

```
SCALEFACTORS
xfact xoff
yfact yoff
```

Initialization sequence to put the terminal into graphics mode.

Default: `no init. sequence`

```
VECTOR_INIT_MODE
sequence
```

Put the terminal into vector graphics mode.

Default: `sequence =  <GS>`

```
VECTOR_MODE
sequence
```

Alpha graphics mode.

Default: `sequence =  <US>`

```
ALPHA_MODE
sequence
```

Point graphics mode.

Default: `sequence = <FS>`

```
POINT_MODE
sequence
```

B GENERIC TERMINAL FILE EXAMPLES

B.1 VT640

The following is an example of a VT640 generic terminal characteristics file.

```
TERMINAL_TYPE
TEK4010
RANGE
0. 640. 0. 479.
DOTS_ON
<ESC>/0d
DOTS_OFF
<ESC>/1d
DOTS_COMPLEMENT
<ESC>/2d
GRAPHICS_CLEAR
<ESC><FF>
TRANSPARENT_CLEAR
<CTRLX><ESC>[2J<ESC>[1;1H
ENTER_TRANS_MODE
<GS><CAN>
! 1.6         = 1024./640.
! 1.625       = 780./480.
! 0.625       = 640./1024.
! 0.615384615 = 480./780.
CROSSHAIRS
<GS>
<ESC>/f<ESC><SUB>
1.6    0.0
1.625  0.0
0.625  0.6
0.615384615   0.6
SCALEFACTORS
 1.6   0.5
 1.625 0.5
VECTOR_MODE
<GS>
ALPHA_MODE
<US>
POINT_MODE
<FS>
```

B.2 CIT467

The following is an example of a CIT467 generic terminal characteristics file.

```
TERMINAL_TYPE
TEK4010
RANGE
0.0  571.0  0.0  479.0
SPECIAL
<ESC>1<ESC>0 !VEC 0,0,0,0 !BYE
DOTS_ON
<ESC>1<ESC>o
DOTS_OFF
<ESC>1<ESC>h
GRAPHICS_CLEAR
<ESC>2<ESC>7<CTRL]><ESC><FF><ESC>2<ESC>8
TRANSPARENT_CLEAR
<ESC>2<CTRLX><ESC>[2J<ESC>[1;1H
ENTER_TRANS_MODE
<ESC>2
! 1.789807692 = 1023.77/572.
! 1.640647182 = 785.87/479.
SCALEFACTORS
 1.789807692  0.0
 1.640647182  0.0
! 1.792942207 = 1023.77/571.
! 1.640647182 = 785.87/479.
! 0.557742462 = 571./1023.77
! 0.609329483 = 479./786.11
CROSSHAIRS
<CTRL]>
<ESC>/f<ESC><CTRLZ>
 1.792942207  0.0
 1.640647182  0.0
 0.557742462  0.6
 0.609329483  0.6
COLOURS
 0,8
 <ESC>1<ESC>h
 <ESC>1<ESC>o
 <ESC>1<ESC>i
 <ESC>1<ESC>l
 <ESC>1<ESC>j
 <ESC>1<ESC>m
 <ESC>1<ESC>n
 <ESC>1<ESC>k
 <ESC>1<ESC>o
VIDEO_OFF
<ESC>1<ESC>V<ESC>2
VIDEO_ON
<ESC>1<ESC>2
VECTOR_MODE
<ESC><GS>
```

B.3 VT241

The following is an example of a VT241 generic terminal characteristics file.

```
TERMINAL_TYPE
REGIS
RANGE
0.0  639.0  0.0  479.0
DOTS_ON
W(V)
DOTS_OFF
W(E)
DOTS_COMPLEMENT
W(C)
GRAPHICS_CLEAR
<ESC>\<ESC>[2J<ESC>[1;1H
TRANSPARENT_CLEAR
<CAN><ESC>[2J<ESC>[1;1H
ENTER_TRANS_MODE
<ESC>\
CROSSHAIRS
P
R(P(I))
 0.0 0.0
 0.0 0.0
 0.0 0.0
 0.0 0.0
COLOURS
 0,7
+W(I(D))
+W(I(W))
+W(I(R))
+W(I(G))
+W(I(B))
+W(I(Y))
+W(I(C))
```

```
+W(I(M))
```

B.4 KERMIT

The following is an example of a KERMIT gereric terminal emulator, with VGA colour.

```
DOTS_ON
^]<ESC>/0d
DOTS_OFF
^]<ESC>/1d
DOTS_COMPLEMENT
^]<ESC>/2d
GRAPHICS_CLEAR
^]^X<ESC>7^]<ESC><FF>^]^X<ESC>8<US>
TRANSPARENT_CLEAR
^X<ESC>[2J<ESC>[1;1H
VECTOR_INIT_MODE
<ESC>[?38h
VECTOR_MODE
<GS>
ENTER_TRANS_MODE
<US>
CROSSHAIRS
<GS>
<ESC><CTRLZ>
1.6    0.0
1.625  0.0
0.625  0.6
0.615384615   0.6
SCALEFACTORS
1.6   0.5
1.625 0.5
COLOURS
0,8
<ESC>[30m
<ESC>[37m
<ESC>[31m
<ESC>[32m
<ESC>[34m
<ESC>[33m
<ESC>[36m
<ESC>[35m
<ESC>[38m
```