GPL

GPL is a collection of Fortran graphics routines, implemented locally at UD, that may be invoked using the CALL statement. Some routines have **optional arguments** (denoted by underlining) An optional argument may be omitted if the **default** value is adequate. (Optional arguments underlined as a group should be included or deleted as a group).

CONTENTS

COMPILING

To compile a program that contains calls to these, you have the alias (created in your initial setups for this course) `f95g` which will compile the code using the necessary links:

`f95g` *`program_file`*`.f95`

followed by issuing the usual command `a.out` to run the program. The `f95g` alias includes the necessary options to find the **`GPL`** module.

Startup and Shutdown

To access GPL routines in a Fortran program, you must include the following statement as the first statement after the program or subroutine statement (before the `IMPLICIT NONE` statement):

```
USE GPL
```

The `USE` statement names a `MODULE` containing library information.

Before calling any other GPL routines, you need to initialize graphics devices:

```
CALL OpenGP (aspect, devices)
```

`aspect` `REAL` aspect ratio, height-to-width, of drawing area, must be 1.0 or less. **Default**: `aspect=1.0` (square drawing area). If `aspect` = 0.5, for example, the graph box will be twice as wide as it is high. Setting `aspect` to 0.8 mimics the shape of a standard piece of paper.

`devices` Integer control scalar. If you only want the screen plot and not the PostScript file, set `devices=1`. If you only want the PostScript file and not the screen plot, set `devices=2`. **Default** is to create both.

This should be the first graphics call in any Fortran program that uses GPL. It performs device initialization: putting up the X-terminal window in which the graphs are drawn and initializing the PostScript file to which a printable form of the graphs can be written. Typical usage has no arguments.

```
CALL CloseGP ( psfile )
```

`psfile` `CHARACTER` string containing a new file name for the graphics output file. **Default** is to call the file `gpl.ps`. **Caution:** GPL destroys any old versions of `gpl.ps` and creates a new one for any call to `OpenGP` with devices at default or 2, regardless of the presence or value of `psfile` on the subsequent `CloseGP` call. The optional `psfile` argument acts by renaming the `gpl.ps` file that has just been created.

This closes the graphics window, finishes the graphics file and optionally renames it. The most common usage is to call it with no arguments. Use of this before the final use of GPL graphics in a program requires that the OpenGP call be repeated before any further graphics, resulting in a new graphics window and new graphics output file. The optional argument is used only to rename the PostScript file.

Two-Dimensional Data Graphs (X-Y graphs)

This group consists of four routines: GPLR, GPLV, GPLV2 and GPLF. The first three can produce both connected curve plots and scatter plots, while GPLV and GPLV2 can mix scatter plots and connected curves. GPLR can also produce simple regression plots, while GPLF can produce regression plots with higher-order polynomials.

GPLR

CALL GPLR (x, y, mode, xlabel, ylabel, toplabel, axis, xaxis, yaxis)

GPLR is the simplest GPL x-y plotting routine for generating a connected curve plot or a scatter plot. GPLR works if you have only one set of points to be connected or marked, or if you have multiple sets of points that share a common set of x values.

x Real one-dimensional array containing the x-coordinates of a list of data points. If you are putting more than one curve on a graph, then this set of x-coordinates will be used repeatedly, once for each curve.

y Real one- or two-dimensional array containing the y-coordinates corresponding (in sequence) to the x values in x. If y is a one-dimensional array, then it should be the same length as the array x, and one curve (or scatterplot) will be generated.

If y is two-dimensional, then two or more curves or scatterplots will be generated. Then y should have a shape in which the first index corresponds to the values in x and the second index is the curve number, y(*size_of_x,number_of_curves*).

mode Integer control scalar with three possible values:

0 Plot the data as a series of connected points. **Default** value.

1 Plot a marker point for each datapoint, creating a scatterplot.

2 Plot a stepped curve, consisting of a horizontal line over the interval between each datapoint. Often used when a single y value applies equally to an entire interval, such as in monthly data.

If more than one curve is plotted on a single graph (y is two-dimensional), then GPLR uses a different dashpattern for each curve. Up to 8 dashpatterns are available, and then the sequence will start repeating itself: solid, long dash, short dash, long dash-short dash, dotted, dot-medium dash, tiny dash, long dash-three dots. If more than one scatterplot is plotted on a single graph, GPLR uses four different markers in sequence:

✶ o x +

xlabel, ylabel, toplabel Character string labels for the bottom, left, and top of the graph. (They may include embedded commands, discussed in the vector primitives section, to use symbols, superscripts, or subscripts.) Each label should be no longer than about 30 characters. **Default** is to plot no label for arguments that are not included.

axis Integer control number that sets some options about the appearance of the plot. This variable is specified as a sum of options, meaning that you add up the numbers associated with each option you wish to turn on, and specify the sum as the actual argument. **Default:** 0, implying that none of these options are turned on.

- 1 Invert axes: plot x on the vertical axis and y on the horizontal axis, often used when x is usually perceived vertically, such as when it is a depth, height, or latitude. **Note:** xaxis, yaxis, xlabel, ylabel, and axis options 2, 4, 16, and 32 are all tied to x and y, not to horizontal and vertical. Using axis option 1 switches the horizontal and vertical sense of all of these together.
- 2 Use a logarithmic scale for the x axis. The range of data for x must be fit entirely within positive numbers for this to work.
- 4 Use a logarithmic scale for the y axis. Positive numbers only.
- 16 Plot a background grid for every major tick of the x axis.
- 32 Plot a background grid for every major and minor tick of the y axis.

xaxis, yaxis Arrays of real numbers that can be used to control the starting point, ending point, and tick sizes of each axis. If included, each consists of two to four REAL numbers with the following meanings:

1st: Minimum value shown on the axis (left or bottom).
2nd: Maximum value shown on the axis (right or top).
3rd: Spacing of major (labeled) tick marks. (Optional)
4th: Number of minor tick marks between each major tick mark. (Optional, and may be included only if 3rd value is included.)

Default is that GPL will calculate reasonable values, sticking to round numbers and powers of ten as much as possible.

Generic argument possibilities. The required input arrays x and y can be KIND=4 (default precision) or KIND=8, so long as both are the same REAL KIND. (If xaxis or yaxis are included, they must be the same REAL KIND as x and y.) All integer and character arguments must be default KIND. A one-dimensional array y can used instead of a y whose shape is (:,1).

GPLR Examples:

```
CALL GPLR ( x=time(1:30), y=speed(1:30) )
```

Plot a connected curve of `speed` versus `time`, with 30 points on the curve. Lack of optional arguments implies: connected curve (not scatterplot), no axis labels, and `speed` is on the vertical axis and `time` on the horizontal axis.

```
CALL GPLR ( x=height(1:10), y=temperature(1:10), axis=1 )
```

Plot a connected curve of `temperature` versus `height`, with 10 points on the curve. The axis option makes `height` the vertical axis variable and `temperature` the horizontal axis variable. Lack of other optional arguments implies: connected curve (not scatterplot) and no axis labels.

```
CALL GPLR ( x=area(1:20), y=pop(1:20,1:2), mode=1, &
     xlabel='County area', ylabel='County population' )
```

Plot a scatterplot (`mode=1`) of `pop versus area`, in which two different `pop` numbers will be plotted (with different symbols) for each of 20 `area` numbers. Descriptive axis labels are supplied for the horizontal and vertical axes, but not for the top.

```
CALL GPLR ( x=hour(1:72), y=temperature(1:72), &
     xaxis=(/0.0, 72.0, 12.0, 5.0/), yaxis=(/0.0, 40.0/) )
```

Plot a connected curve plot of `temperature` versus `hour`. The x axis will go from 0 to 72 hours, with a labeled tick every 12 hours and 5 minor ticks between each labeled tick (i.e., a tick for each two hours). (Left to itself, GPL would never choose ticks as multiples of 12, so hourly data are a good example of when user control of the axis is helpful.) The y axis will go from 0 to 40, but GPL will pick the major and minor tick intervals.

GPLV

CALL GPLV (x, y, curves, mode, xlabel, ylabel, toplabel, axis
xaxis, yaxis)

GPLV can reproduce the functions of GPLR, but it also allows more control. One may have a different number of points for each curve. One can choose the mode (scatter, curve, or stepped plot) for each curve individually, and choose the dashpattern or scatterplot marker as well.

x Real one-dimensional array containing the x-coordinates of each data point. Putting more than one curve on a graph requires a complete set of x-coordinates for each curve, not just one set to be used repeatedly.

y Real one-dimensional array, the same size as x, containing the y-coordinates corresponding to the x-coordinates in x.

curves Integer one-dimensional array containing the number of points in each curve. For example, if one wished to plot three curves of length 50, 10, and 2 respectively, then curves would be an integer array of size 3 containing the three numbers 50, 10, and 2, and the length of x and y would each be 62. In general *SUM*(curves) should equal *SIZE*(x) and *SIZE*(y).

mode Integer one-dimensional array containing a control scalar for each curve. The size of mode is the same as the size of curves. The codes are:

–1 to –5 Mark each point with a marker, controlled by codes:

–1 ✶ –2 **X** –3 **O** –4 **+** –5 **.**

0 Connect the points in a step plot line.

1 to 8 Connect the points directly, using the code to choose the dashpattern. Dashpatterns are chosen by the index numbers in the same sequence used by GPLR: solid, long dash, short dash, long dash-short dash, dotted, dot-medium dash, tiny dash, long dash-three dots.

Option: Specifying sm integer scalar as mode will cause that mode to be used for all the curves.

Default: The connected curve modes, 1 through 8, are used in sequence.

xlabel, ylabel, toplabel Character string labels for the bottom, left, and top of the graph. **Default** is to plot no label if corresponding argument is not included.

axis Integer control scalar specified as a sum of options. **Default:** 0.

1 Invert axes: plot x on the vertical axis and y on the horizontal axis.
2 Use a logarithmic scale for the x axis. Positive numbers only.
4 Use a logarithmic scale for the y axis. Positive numbers only.
16 Plot a background grid for every major tick of the x axis.
32 Plot a background grid for every major and minor tick of the y axis.

xaxis, yaxis Arrays of real numbers that can be used to control the axes. See GPLR for longer descriptions.

1st: Minimum value shown on the axis.
2nd: Maximum value shown on the axis.
3rd: Spacing of major (labeled) tick marks. (Optional)
4th: Number of minor tick marks between each major tick mark. (Optional, and may be included only if 3rd value is included.)

Default is that GPL will calculate reasonable values.

Generic argument possibilities. The required input arrays x and y can be KIND=4 (default precision) or KIND=8, so long as both are the same REAL KIND. (If xaxis or yaxis are included, they must be the same REAL KIND as x and y.) All integer and character arguments must be default KIND. A scalar value for curves or mode can be substituted for a one-dimensional array of size (1).

GPLV Examples:

```
CALL GPLV ( x=time(1:30), y=speed(1:30), curves=(/20,10/) )
```

Plot two connected curves of speed versus time, with the first 20 points on the first curve and the remaining 10 points plotted as a second curve with a different dashpattern. Lack of optional arguments implies: connected curve with solid line for the first curve and dashed line for the second, no axis labels, and speed is on the vertical axis and time on the horizontal axis.

```
CALL GPLV ( x=height(1:100), y=temperature(1:100),
     curves=sounding_sizes, mode=1, xlabel='Height (m)', &
     ylabel='Temperatures (C)', axis=1 )
```

Plot a set of connected curves of temperature versus height, with 100 points available. How these points are split among the curves will depend on the value of sounding_sizes which is sent in as the actual argument for curves. For example, if sounding_sizes=(/30,40,30/), then there will be three curves, with 30 points in the first, 40 in the second, and 30 in the third. All curves will be plotted as solid lines (mode=1). The axis option makes height the vertical axis variable and temperature the horizontal axis variable. Descriptive axis labels are provided. Notice that xlabel is tied to x, not to the horizontal axis—using axis=1 will switch both.

GPLV2

```
CALL GPLV2 ( xl, yl, xr, yr, lcurves, rcurves, lmode, rmode,
             botlabel, llabel, rlabel, toplabel, axis, xaxis,
             laxix, raxis )
```

GPLV2 reproduces all of the functions of GPLV, and in addition allows curves to have separately scaled and labeled right and left vertical axes. Most of these arguments are just left and right versions of the same arguments as in GPLV.

xl, yl Real one-dimensional arrays containing x-coordinates and y-coordinates of all points to be plotted with respect to the left-side y-axis. These two arrays need to be the same size.

xr, yr Real one-dimensional arrays containing x- and y-coordinates for data to be plotted against the right-side y-axis. These two arrays need to be the same size as each other; they need not be the same size as xl and yl.

lcurves, rcurves Integer arrays indicating how many points are in each curve, equivalent to curves in GPLV.

lmode, rmode Integer arrays containing a control scalar for each curve. The codes, the option of specifying one integer scalar as mode for all curves, and the **default** of using connected curve modes, 1 to 8, in sequence, are also the same as for GPLV.

–1 to –5	Mark each point with a marker: –1 ✶ –2 **X** –3 **O** –4 **+** –5 **.**
0	Connect the points in a step plot line.
1 to 8	Connect the points directly; mode is dashpattern number.

botlabel, llabel, rlabel, toplabel Character string labels for the bottom, left, right, and top of the graph. **Default** is to plot nothing for an argument that is not included.

axis Integer control scalar specified as a sum of options. (Axis inversion is not available for this routine.) **Default**: 0.

2	Use a logarithmic scale for the x axis.
4	Use a logarithmic scale for the left vertical axis.
8	Use a logarithmic scale for the right vertical axis.
16	Plot a background grid for every major tick of the x axis.

xaxis, laxis, raxis Arrays of real numbers that can be used to control the axes. See GPLR for longer descriptions.

1st: Minimum value shown on the axis.
2nd: Maximum value shown on the axis.
3rd: Spacing of major (labeled) tick marks. (Optional)
4th: Number of minor tick marks between each major tick mark. (Optional, and may be included only if 3rd value is included.)

Default is that GPL will calculate reasonable values

Generic argument possibilities. The required input arrays xl, yl, xr and yr can be KIND=4 (default precision) or KIND=8, so long as all four are the same REAL KIND. All integer and character arguments must be default KIND. A scalar value for curvesl, curvesr, model, or moder can be substituted for a one-dimensional array of size (1).

GPLF

CALL GPLF (x, y, degree, coefs, xlabel, ylabel, toplabel, pcoef, xaxis, yaxis)

GPLF fits a polynomial curve to a scatter plot, using the least-squares method to fit the curve. The ten arguments to GPLF are:

x Real one-dimensional array containing x-coordinates of the data points.

y Real one-dimensional array containing the y-coordinates corresponding (in sequence) to the x values in x. It should be the same size as x.

degree Integer scalar indicating the degree of the polynomial fitted.
0 Plot a horizontal line at the mean of the y values.
1 Plot a regression line through the scatter plot.
2–10 Calculate and plot coefficients for a polynomial fit of the given order. For example, if degree=3, then calculate the coefficients c_0, c_1, c_2, and c_3 for the cubic polynomial regression equation

$$\hat{y} \doteq c_0 + c_1 x + c_2 x^2 + c_3 x^3$$

and plot the resulting curve along with the scatterplot. **Default** value is to produce a regression line, degree=1.

In general, if d is the degree, then the regression equation is

$$\hat{y} \doteq \sum_{k=0}^{d} c_k x^k$$

and d can be any number up to n – 1, where n is the number of data pairs in x and y. In practice, the procedure used for calculating the set of coefficients c_k is not numerically reliable for d > 10, regardless of the value of n.)

coefs Real, one-dimensional array of size at least degree+1, intent out. If provided, then GPLF will return the values of the coefficients c_k, in order c_0 through c_d, so that later parts of the program may use these for calculations. **Default** is to not return the coefficients. The coefficients will be calculated within GPLF, but the calling program may have no use for them

xlabel, ylabel, toplabel Character string labels for the bottom, left, and top of the graph. **Default** is to plot no label for any argument that is not included.

pcoef Character control scalar that determines where and if the polynomial regression coefficients will be plotted on the figure.

- T Plot regression equation on **t**op of the graph (replaces `toplabel` if provided).
- I Plot regression equation **i**nside the graph box near the top. This will not overwrite or replace the top label, but may interfere with some of the graphed data or curve. This is the **default**.
- B Plot regression equation inside the graph box near the **b**ottom.
- N Plot **n**o regression equation.

Both lowercase and uppercase versions of these will be tested. Only the first character of a longer string will be tested, so entire words may be used so long as the initial matches the above set.

xaxis, yaxis Arrays of real numbers that can be used to control the axes. See `GPLR` for longer descriptions.
1st: Minimum value shown on the axis..
2nd: Maximum value shown on the axis, to be plotted right or top.
3rd: Spacing of major (labeled) tick marks. (Optional)
4th: Number of minor tick marks between each major tick mark. (Optional, and may be included only if 3rd value is included.)
Default is that GPL will calculate reasonable values.

Generic argument possibilities. The required input arrays `x` and `y` can be `KIND=4` (default precision) or `KIND=8`, so long as both are the same `REAL KIND`. (If `xaxis` or `yaxis` are included, they must be the same `REAL KIND` as `x` and `y`.) All integer and character arguments must be default `KIND`.

GPLF Examples:

```
CALL GPLF ( x=height(1:10), y=temperature(1:10), degree=1)
```

Plot a regression plot of `temperature` versus `height`, with 10 points marked with scatterplot symbols and a regression line drawn through the data cloud. By default, the regression equation will also be plotted inside the graph box

```
CALL GPLR ( x=area(1:20), y=pop(1:20), degree=3, coefs=c,
     pcoefs='No', &
     xlabel='County area', ylabel='County Population' )
```

Plot a scatterplot of `pop` versus `area`, with a cubic polynomial (`degree=3`) fit to the data and plotted. Coefficients will not be plotted (`pcoefs='No'`) but will be returned to the program in the array `c`, which must be of size at least 4. Descriptive axis labels are supplied for the horizontal and vertical axes, but not for the top.

Three-dimensional Plots

GPL contains a few routines for doing contouring and shading of a third dimension rising above or sinking below the plane of the graph. These routines are as completely developed as the xy plotting routines, in the sense that few options are available and customization of your output graph might require modifying the code. Contour line labeling is done poorly by these routines.

The traditional way of representing a third dimensional field is to use isolines or isopleths, but these are commonly called contours because of their early use in topographic maps. For contouring, GPL includes GPLConR, which does contouring of **randomly distributed** data within a boundary, and GPLConG, which contours data that can be given as values along an evenly-spaced **rectangular grid**.

The widespread availability of grayscale and color printing, as well as of only creating graphics for on-screen viewing, has led to more extensive use of shading to represent the third dimension. As with contouring, GPL contains routines for gridded and randomly distributed data: GPGShade and GPRShade. Only GPGShade is presented here—GPRShade contains documentation in comments within source codes in ~hanson/gpl/src.

GPLConR

CALL GPLConR (x, y, z, xb, yb, toplabel, contour_options)

x, y Real one-dimensional arrays containing x and y coordinates of each raw data point.

z Real one-dimensional array containing coordinate of the "dependent" third-dimension variable that will be contoured.

(x, y, and z should all be the same size.)

xb, yb Real one-dimensional arrays of the same size specifying x and y coordinates of a sequence of points that will specify a boundary. **Default**: no boundary drawn; contours go to the convex hull of the data.

toplabel Character string that will be used to annotate the top of the graph. **Default:** no label plotted.

contour_option*s* Integer control variable, given as 0 or sum of the following options. **Default**: 0.

1 Suppress contour line labeling.
2 Suppress line thickness differences between major and minor lines.
4 Add annotation of local maxima and minima.

Generic argument possibilities. x, y, z, xb, and yb can be of KIND=4 (default precision) or KIND=8, so long as all five are of the same REAL KIND. All integer and character arguments must be default KIND.

GPLConG

Note the difference in the x and y arrays: only the grid values need to be included—not a complete set of x and y values for each z point.

CALL GPLConG (x, y, z, label, contour_options, label2, label3)

The seven arguments to **GPLConG** in sequence, are:

x, y Real one-dimensional arrays containing the x and y coordinates of the grid data points.

z Real two-dimensional array containing values of the "dependent" third-dimension variable that will be contoured.

Assume nx=SIZE(x) and ny=SIZE(y). Then without specifying contour_options, the shape is (nx,ny), in which case variations of the first index correspond with changing values of x, values of the second index correspond with changing values of y. If contour_options 8 is specified, this array is shape (ny,nx) and the sense of the indexes is switched accordingly.

label Character string that will be used to annotate the top of the graph. **Default:** no label will be plotted.

contour_options Integer control variable. 0 or sum of options. **Default**: 0.

1 Suppress contour line labeling.
2 Suppress line thickness differences between major and minor line.
4 Add annotation of local maxima and minima.
8 z is inverted from standard form (x is vertical, y is horizontal).
64. Add right-side axis label showing contour minimum, maximum and increment.
256. Plot world map outline on top of contours. Assumes that the domain of the contours is cylindrical equidistant projection with left and right borders at 180° longitude and bottom and top at the poles.

label2 Character string that will be used as second line of annotation at top. **Default:** no label will be plotted. If label is not present, label2 will not be plotted.

label3 Character string that will be used as third line of annotatioon at top. **Default:** no label will be plotted. If label2 is not present, label3 will not be plotted.

Generic argument possibilities. x, y, and z can be of KIND=4 (default precision) or KIND=8, so long as all three are of the same REAL KIND. All integer and character arguments must be default KIND.

GPGShade

Gridded shading routine, for points in a rectangular grid. Data are assumed to represent values on a point grid. This routine shades little rectangles based on the average of the four corner values of that rectangle.

```
CALL GPGShade ( x0, y0, x1, y1, v, label, shade_options,
                threshholds, colortable, label2, label3, xb, yb )
```

The arguments to **GPLConG** are:

x0, y0 Real scalars indicating the x and y coordinates of the lower left corner of the grid (can be interpreted as x_{min} and y_{min}).

x1, y1 Real scalars indicating the x and y coordinates of the upper right corner of the grid (can be interpreted as x_{max} and y_{max}).

v Real two-dimensional array containing values of the "dependent" third-dimension variable that will be contoured. Without specifying Shade_Options, the shape is interpretable `(nx,ny)`, (number of points in x direction and number of points in y direction). If `contour_options` 1 is specified, this array is shape `(ny,nx)` and the sense of the indexes is switched accordingly.

label Character string that will be used to annotate the top of the graph. **Default:** no label will be plotted.

shade_options Integer control variable. 0 or sum of options. **Default**: 0.

- 1 Invert the x and y sense of the dimensions on `v`, such that first index represents vertical variation and second index represents horizontal variation.
- 2 Shade in a page-filling size, assuming that x and y are not commensurate. Shaded box will be a square in the default viewport.
- 64 Place a right-side label showing the minimum and maximum values encountered in the data.
- 256 Plot world map outline on top of shading. Underdeveloped option assumes that the domain is cylindrical equidistant projection (i.e., unprojected) with left and right borders at 180° longitude and bottom and top at the poles.

threshholds Real one-dimensional array. If specified, this contains the "cutoff" points between colors. This contains one more threshhold than the number of colors that will be plotted, as it contains endpoints (subminimum and supermaximum) that will be plotted on the legend colorbar. Must be specified in monotonic lowest to highest order. **Default**: If absent, reasonable, evenly spaced, round-number threshholds will be calculated from the minimum and maximum of `v`.

colortable Integer scalar or one-dimensional array.

If scalar or array of size 1, this chooses a preset color table from among a set currently listed within GPL. Except for the first, default colortable, these will be highly device dependent and subject to change with time as drivers vary. On PostScript drivers (obsolete) these can be described in the following color ranges, and number of different colors.

0 Grayscale/monochrome, white low and black high, 26.
1. Grayscale/monochrome, black low and white high, 26
2. Pale rainbow, purple low and red high, 28.
3. Saturated rainbow, purple low and red high, 31.
4. Green to red, 19.
5. Widely separated rainbow, 10.
6. Red to blue, 20.
7. Yellow to blue, 16.
8. Blue to red, white in middle, 17.

Note that the number of threshholds can be no more than the number of colors plus one, or colors will repeat. If the number of threshholds is less than the number of available colors, then colors will be skipped to hit the endpoints of the colortables.

If array of size greater than 1, then this is a user-specified colortable of values that will control shading via the `color` option in the GPL primitive routine `set_poly`. A driver color table for the intended output device is required for constructing such a table, but users may look at the colortable specifications for the presets in `gpl_colortable` to get ideas.

Default: If absent, or if an undefined preset is requested, colortable 0 (grayscale, white to black) will be used.

label2 Character string that will be used as second line of annotation at top. **Default:** no label will be plotted. If `label` is not present, `label2` will not be plotted.

label3 Character string that will be used as third line of annotatioon at top. **Default:** no label will be plotted. If `label2` is not present, `label3` will not be plotted.

xb, yb Real, dimension(4), set of latitudes and longitudes specifying corners of a map, preferably within the contiguous United States. Specifying `shade_options 256` with these produces a shaded state map. Extremely simplistic—use with high caution and low expectation.

Generic argument possibilities. All real variables can be of `KIND=4` (default precision) or `KIND=8`, so long as all are of the same `REAL KIND`. All integer and character arguments must be default `KIND.` If a preset colortable is being chosen, it may be an integer scalar or an integer array of size 1.

Vector Primitives

Vector primitives produce the most basic graphics objects. This set of routines can draw straight lines, sequences of connected straight lines, characters and character strings, and shaded polygons. Also included are routines which scale a diagram, move around on the drawing space without drawing anything, choose a "pen" (color, thickness, and dash pattern of a line), choose a font, and choose a shading pattern or color for polygons.

Generic possibilities. In all of these routines, `REAL` arguments may be `KIND=4` (default precision) or `KIND=8`, so long as all `REAL arguments` are of the same `KIND`. All `INTEGER` and `CHARACTER` arguments must be of the default `KIND`. A two-dimensional array argument of shape `(:,1)` may be submitted as a one-dimensional array, and a one-dimensional array of size `(1)` may be submitted as a scalar.

Frame Setup

All of the startup and shutdown considerations discussed on page 2 of this handout still apply to the vector primitives. In addition, two routines are needed to set up and clear each frame of graphics produced using the vector primitives. A "frame" is a single graph or page of drawing.

`CALL Frame_Start ( xmin, xmax, ymin, ymax )`

This is the first call of any new graphics frame.

`xmin, xmax, ymin, ymax` `REAL` scalar arguments that establish a scale transformation between the user's coordinate system and the drawing space. `xmin` and `xmax` become the user coordinates of the left side and right side of the drawing area, respectively, and `ymin` and `ymax` become the user coordinates of the bottom and top of the screen. **Default**: each pair will be set to `(-1.0, 1.0)`.

`CALL Frame_End (wait, clear)`

This is the last call of a graphics frame.

`wait` `INTEGER` code: if any value other than 0, the program will pause so that the user can view the results before proceeding with the next graph. If 0, there is no wait. **Default**: `wait=1` (wait for a mouse click on the graphics window before proceeding).

`clear` `INTEGER` code: any value other than 0, clear the graphics window before the next frame begins. If 0, do not clear, but overlay the next graph on the current one. **Default**: `clear=1` (clear the window).

Drawing Lines and Curves

CALL Set_Pen (dash, color, width)

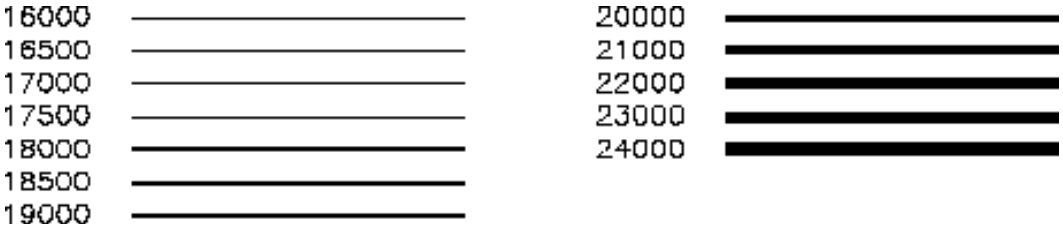
Set the conditions for subsequent calls of the Pen_Draw or Pen_RDraw. These conditions stay set until subsequently changed. Calling Set_Pen with no arguments resets all three arguments to their default values.

dash INTEGER code to select a dash pattern. **Default**: 0 (solid line).

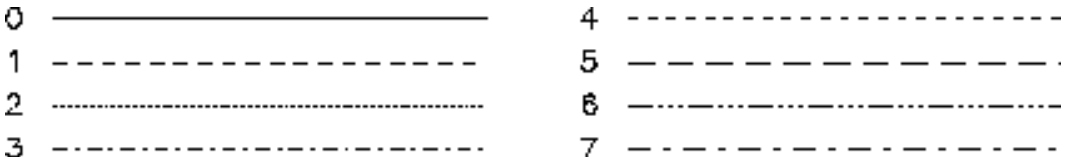
color INTEGER code to select a color. Results vary with device **Default**: 0 (device default—black on typical printers).

width INTEGER code to select line width. **Default**: 16000 (a minimum)

Line Width Codes **(**width ***in*** Set_Pen**)**



Dash Pattern Codes **(**dash ***in*** Set_Pen**)**



CALL Pen_Draw (x, y)

The most basic vector graphics element: draw a straight line from the current position (established via a Pen_Move call or a previous Pen_Draw call) to the position(s) given by the arguments. Several variations in arguments are allowed.

If **x** and **y** are both REAL scalars, then one line will be drawn from the current position to the position (x,y) in the user's coordinate system.

If **x** and **y** are both REAL one-dimensional arrays of the same length n, then a sequence of lines will be drawn, first to the position (x_1,y_1) and then connecting all the remaining points in the array ending with (x_n,y_n).

If **y** is absent then **x** must be a REAL two-dimensional array on which at least one dimension must be 2. In this case, (x,y) pairs of data are assumed included together in the array. For example, CALL Pen_Draw (T(1:N,1:2)) is equivalent to CALL Pen_Draw (T(1:N,1), T(1:N,2)).

CALL Pen_RDraw (x, y)

Similar to Pen_Draw, except that coordinates given are not absolute coordinates, but rather relative coordinates. For example, CALL Pen_RDraw (1.0, 0.5) draws a line a distance 1.0 up and 0.5 right from wherever the pen currently resides. Argument options are the same as for Pen_Draw.

Moving without Drawing

CALL Pen_Move (x, y)

Move the pen to a location without drawing a line. This is usually needed for the first point of a line or curve. Arguments are REAL scalars in absolute user coordinates.

CALL Pen_RMove (x, y)

Relative move: move the pen a distance **x** horizontally and **y** vertically in the user coordinate system.

Drawing Characters and Character Strings

CALL Set_Char (csize, justify, font, base_angle, path)

Sets the conditions for subsequent calls to Char_Draw. Calling Set_Char with no arguments will reset all five to their default values.

csize REAL one-dimensional array of size 2 which contains the size of the "box" in which each character is drawn. For example,
CALL Set_Char (csize=(/ 0.05, 0.07 /))
requests that characters each occupy a box 0.05 wide and 0.07 high in the user coordinate system. **Default**: (/ 0.02, 0.02 /) (usually too small).

justify CHARACTER one-dimensional array of size 2 ... or
INTEGER one-dimensional array of size 2

Establishes the justification of the text strings according to the table below—first element of the array is horizontal justification and second element is vertical justification. If a character array is used, only the first character of each element is tested, and case will not matter.

Horizontal Justification		Vertical Justification	
1 or 'L'	left	1 or 'B'	bottom
2 or 'C'	center	2 or 'C'	center
3 or 'R'	right	3 or 'T'	top

For example: CALL Set_Char (justify=(/ 1, 3 /)) is the same as CALL Set_Char (justify= (/ 'L', 'T' /)), and both request that the subsequent character strings be drawn left-justified and top-justified (hanging) relative to the starting position. **Default**: (/ 1, 1 /) which is (/ 'L', 'B' /).

font INTEGER code: see the table. **Default**: font=1, a block simplex font.

base_angle REAL scalar. Angle in degrees, relative to horizontal, of the baseline along which characters will be drawn. For example, base_angle=90.0 would request a vertical upward baseline, so characters would be readable from the right side of the page. **Default**: base_angle=0.0 (draw the characters on a horizontal line).

path INTEGER code which chooses the orientation of the characters relative to the baseline. **Default** (path=1) is seldom changed. Possible values:

1 – right (left to right along the character base line)
2 – down (top to bottom, perpendicular to base line)
3 – left (right to left along the base line)
4 – up (bottom to top, perpendicular to base line)

Font Codes **(**font ***in*** Set_Char**)**

CALL Char_Draw (string, x, y)

Draws a character string on the plot, with font specifications given in the most recent call to Set_Char.

string CHARACTER string of any length to be drawn on the plot.

x, y REAL scalars that establish the starting point for the character string. If both are deleted, the current pen position (as last set, for example, by Pen_Move or Pen_Draw) is used as the starting point.

Embedded commands in character strings:

Embedded commands all consist of uppercase strings enclosed in square brackets that may be included within the character string argument of Char_Draw. These include:

[BSUP]	[ESUP]	Begin and end superscript mode.
[BSUB]	[ESUB]	Begin and end subscript mode.
[BUND]	[EUND]	Begin and end underline mode.
[FONT=**n**]		Change to font number **n**.
[FONT]		Change back to the original font.

For example,

```
CALL Char_Draw (string='CO[BSUB]2[ESUB] effect ([BSUP]o[ESUP]C)')
```

will produce the following plotted result

CO_2 effect (^oC)

Drawing Shaded Polygons

CALL Set_Poly (edge, color, hatch)

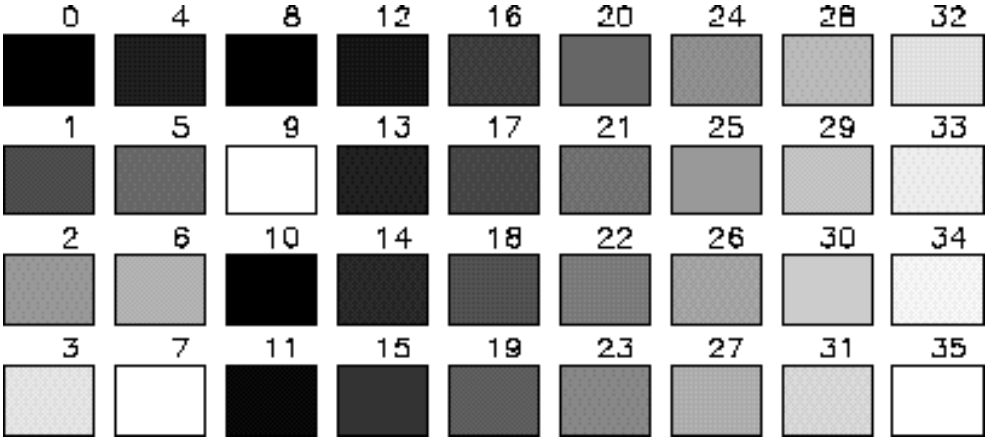
This sets choices of shading, edge drawing, and color (or gray scale) to be used for any subsequent calls to Poly_Draw, Poly_Rect, or Reg_Poly. These choices remain in effect over multiple polygon calls until changed. Calling Set_Poly with no arguments resets all three arguments to their defaults.

edge INTEGER code: if 0, do not draw a line around the boundary (edge) of the polygon. If 1, draw the boundary. **Default**: 1 (draw the boundary).

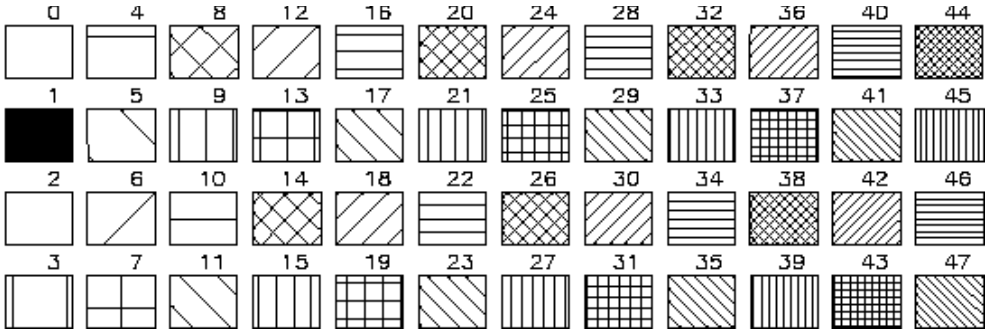
color INTEGER code: again, this is a device-dependent color choice. However, on PostScript laser printers, color choices turn into shades of gray (see the table) so this is useful. **Default**: 0 (leave it blank) unless a hatching pattern is turned on. If a hatching pattern is turned on, the default color is 1 (black). Colors will actually appear as colors in an X-terminal window, so debugging of PostScript output should be done with ghostview.

hatch INTEGER code: see the tables, noting that different hatching patterns will appear in the X-terminal window than in the PostScript output. **Default**: 0 (no hatching).

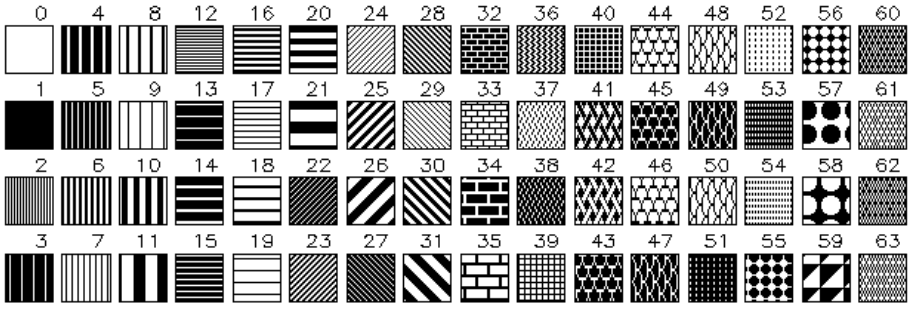
Gray Scale Color Codes—PostScript printers **(**`color` ***in*** `Set_Poly`**)**



Hatching Pattern Codes—PostScript **(**`hatch` ***in*** `Set_Poly`**)**



Hatching Pattern Codes—X window displays **(**`hatch` ***in*** `Set_Poly`**)**



CALL Poly_Draw (x, y)

Draw the polygon whose vertex positions are given in the argument list, applying shading, hatching, or edge as requested by the most recent call to Set_Poly.

x, y REAL one-dimensional arrays of the same length, where that length must be at least 3. These will be assumed to be the x and y coordinates of the polygon vertices. (If y is omitted, then x must be a two-dimensional array on which one of the dimensions is 2. The two-dimensional array then contains both the x and y coordinates of the vertices, as described in Pen_Draw.)

CALL Poly_Rect (x1, y1, x2, y2)

Draw a rectangle, with shading, hatching, and edge as requested by the most recent call to **Set_Poly.** The arguments are REAL scalars, with two different possible meanings.

All four arguments included: **(x1,y1)** are the user coordinates of one corner of the rectangle, and **(x2,y2)** are the user coordinates of the diagonally opposite corner of the rectangle.

Only **(x1,y1)** included: the current pen position is one corner of the rectangle, x1 is the width of the rectangle, and y1 is the height of the rectangle. Numbers can be negative in this case, indicating that width is to the left of the starting point or height is below the starting point.

CALL Reg_Poly (xc, yc, radius, nv, start_angle)

Draw a regular polygon (a polygon that can be inscribed in a circle) with shading, hatching, and edge as requested by the most recent call to Set_Poly.

xc, yc REAL user coordinates of the center of the polygon.

radius REAL radius of the circle in which the polygon will be inscribed.

nv INTEGER number of vertices on the polygon. If omitted or less than 3, a large number of vertices will be chosen in an attempt to draw a circle.

start_angle REAL angle (in degrees) of the first vertex, relative to the center point. **Default**: 0.0 (the first vertex will be at distance radius directly to the right of the center point.

Cue Sheet

Startup and shutdown

CALL OpenGP (aspect, devices=1 for screen only, 2 for PostScript only, **)**
CALL CloseGP

Two-dimensional (XY) plots

CALL GPLR (x=x data array, **y**=y data array, **mode, xlabel, ylabel, toplabel, axis, xaxis, yaxis)**
CALL GPLV (x=x data array, **y**=y data array, **curves, mode, xlabel, ylabel, toplabel, axis, xaxis, yaxis)**
CALL GPLV2 (xl=x data array for left axis, **yl**=y data array for left axis, **xr**=x data for right axis, **yr**=y data for right axis, **lcurves, rcurves=, lmode, rmode, botlabel, llabel, rlabel, toplabel, axis**s, **xaxis, laxis, raxis)**
CALL GPLF (x=x data array, **y**=y data array, **degree**=highest exponent of polynomial to be fit (0–10), **coefs**=array of fitted polynomial coefficients – returned intent(out), **xlabel, ylabel, toplabel, pcoef**=control plotting of polynomial equation, **xaxis, yaxis)**

Common arguments:

xlabel, ylabel, rlabel, llabel, toplabel — character string graph labels
mode, lmode, rmode — choice of connected curve or scatter plot
curves, lcurves, rcurves — integer array giving number of points in each curve
axis — integer sum-of-options variable to choose inverted axes or log scales
xaxis, yaxis, laxis, raxis —control scale and tick labeling for indicated axis.

Three-dimensional (contouring and shading) plots

CALL GPLConR (x=x positions of data points, y=y-positions of data points, **z**=z (plotted variable) data, **xb**=x coordinates of boundary, **yb**=y-coordinates of boundary, **toplabel, contour_options)**
CALL GPLConG (x=x-positions of grid points, **y**=y-positions of grid points, **z**=z (plotted variable) at grid intersections, **label, contour_options, label2, label3)**
CALL GPGShade (x0=minimum x, **y0**=minimum y, **x1**=maximum x, **y1**=maximum y, **v**=plotted variable, **label, shade_options, threshholds**=array of cutoff values for color changes, **colortable**=colors that correspond to values, **label2, label3, xb, yb)**

Common arguments:

toplabel, label, label2, label3 — top-of-graph labels
contour_options — integer array giving number of points in each curve
shade_options — integer sum-of-options variable to choose inverted axes or log scales