8. Graphics Subsystem

The FORTRAN CALCULUS graphics subsystem is a set of utilities which produce graphs in a two stage process, *synthesis* and *production*.

Graph synthesis is the creation and storage of graph information in a data base. The collection of information making up a graph is called a *graph object*, and it is referenced by name. The synthesis process permits ordered information in several graphs to be assembled in a relatively random process concurrently, as it becomes available during computation. The resulting objects contain automatic programs for producing the graphs when they are retrieved by the production utilities. Graph production is the retrieval of graph objects followed by display on a monitor, printing on a graphics compatible printer, or plotting via a pen plotter.

The following utility command-calls are used for graph synthesis:

@PATH - Defines a directory path for the graph database

@GRAPH - Opens a graph object in the graph database

@WINDOW - Defines monitor graphics window, coordinate ranges and scaling

@FRAME - Defines page frame, corresponding to monitor window

@XAXIS - Defines horizontal axis

@YAXIS - Defines vertical axis

@CHRSIZE - Sets character size for printed graphs

@LABEL - Defines and positions text in graph mode

@TEXT - Defines and positions text in text mode

@NUMBER - Defines and positions numeric values in a graph

@XCLABEL - Defines center label for x-axis

@XELABEL - Defines end label for x-axis

@YCLABEL - Defines center label for y-axis

@YELABEL - Defines end label for y-axis

@SETUP - Defines properties of a curve or point set

@CURVE - Enters connected points on a curve function

@SPLINE - Draws a cubic spline curve through the points of a function

@POINT - Enters discrete (unconnnected) points in a point set

@CONTDEF - Defines contour parameters for a set of contours

@MESH - Generates a mesh grid of a contour function

@CONTOUR - Generates contour curves for a group of contours

The following utility command-calls are used for graph production:

@MODES - Defines display modes for graphics & text

@SHOW - Displays a named graph on a graphics monitor

@DRAW - Writes the named graph on a printer file (name.GPR)

@PLOT - Plots a named graph on a pen plotter Graph Synthesis Command-Calls

@CLEAR - Deletes a graph object from the database

8.1. Graph Synthesis Command Calls

@PATH - Definition of database path

Syntax:

@PATH(*path*)

Symbols:

path : character string

This utility defines the storage drive and directory path. On personal computers the drive should be a RAM disk. If this utility is not used, or the operating system does not allow directory structures, the default device will be used for the database.

@GRAPH - Open graph object in database

Syntax:

@GRAPH(*name, type*)

Symbols:

name : character string (8 chars for PC, 6 chars for Cray)

type : character string (4 chars significant)

This utility opens a graph object which will subsequently be referenced by the value of the name parameter. The type parameter is used only for descriptive purposes at this time.

@WINDOW - Definition of monitor graphics window for graph

Syntax:

@WINDOW(*name, lpix, rpix, bpix, tpix, xmin, xmax, ymin, ymax, xorg, yorg, iopt, yoverx, aspect*)

Symbols:

name	: char string	=	name of graph object
lpix	: integer	=	left hand pixel column defining graph window
rpix	: integer	=	right hand pixel column defining graph window
bpix	: integer	=	bottom pixel row defining graph window
tpix	: integer	=	top pixel row defining graph window
xmin	: real*8	=	value of x variable corresponding to *lpix*
xmax	: real*8	=	value of x variable corresponding to *rpix*
ymin	: real*8	=	value of y variable corresponding to *bpix*
ymax	: real*8	=	value of y variable corresponding to *tpix*
xorg	: real*8	=	position of y-axis along x-axis (need not =0)
yorg	: real*8	=	position of x-axis along y-axis (need not =0)
iopt	: integer	=	scaling method option (see below)
yoverx	: real*8	=	number of y-unit changes for a unit change in x value
aspect	: real*8	=	aspect ratio for display monitor (see below)

This utility sets the monitor window for display of the named graph, and also may be alternatively used to specify a print frame (whenever @FRAME is not used, the inch equivalent of the pixel limits are computing using a 9 inch by 6.75 inch frame equivalent to the CRT size for the mode selected). No actual clipping is done in the associated plotting, so plotting may occur outside the plot window.

If *xorg* is set to -999., then no y-axis will be drawn by a call to @YAXIS. If *yorg* is set to -999., then no x-axis will be drawn by a call to @XAXIS.

A border (plot-free) region may be placed around the actual plotting region by choosing *xmin, xmax* and *ymin, ymax* to form a larger window than required for drawing the axes or the plot itself. That is, *xmin, xmax, ymin* and *ymax* need not correspond to the input variables used to define the axes in @XAXIS and @YAXIS nor to the minimum/maximum values of the actual x-y plot.

There are two possible scaling procedures available. The first (*iopt* = 0) is simpler and guarantees that the entire plot is within the viewing region. However, certain distortions may be imposed to achieve this. The second scaling option (*iopt* = 1) requires

careful preparation by the viewer. However, when used properly, it will produce a distortion-free plot.

The user has some "natural" (or "world") coordinates corresponding to the x and y axes of the plot. Normally, these values must be scaled to fit into the desired CRT viewing region. As an example, suppose x varies from 0 to 5000 or from 0 to 1. Recall that there are 640 columns (high resolution) available to define the horizontal axis. If the x coordinate is associated one-to-one with the screen (column) coordinate then in the first case most of the plot will be off the screen and in the second case it will be barely visible since the plot will be only one pixel column wide! The purpose of the scaling procedure is to generate coefficients which may be used to map the user's x-y plot into the specified viewing region on the CRT display.

For any value of *iopt*, the user's x scale is mapped so that the interval from *xmin* to *xmax* corresponds to column *lpix* to *rpix*. When *iopt* = 0, then the interval *ymin* to *ymax* corresponds to rows *bpix* to *tpix*. The question may then arise as to where any distortion may appear. The sources of distortion are twofold. First, the program may need to "squeeze" the plot in the y direction to get it to fit. As an example, suppose the plot window on the screen is 100 pixel columns by 50 pixel column rows. Suppose we wish to plot a square 80 units on a side and specify *xmin* = 1, *xmax* = 100, *ymin* = 1, *ymax* = 100. With *iopt* = 0, we would end up with a plot of a rectangle 80 columns wide and only 40 rows high. This is the first source of distortion. The user must allow enough room in the plot window for the plot, taking into account the number of units changing in the y scale per unit change in the x scale.

The second source of distortion is more subtle. To return to the previous example, the solution would seem to be to simply specify a plot window of 100 columns by 100 rows. The problem is that there still may be distortion associated with what is called "aspect ratio". It is related to the problem of making a square "square". Suppose you want a square one inch on a side. then you need to know how many columns there are in one inch and how many rows there are in one inch. The number of columns need not be the same as the number of rows. The ratio depends on the aspect ratio of the particular CRT, as well as which mode is selected. If you know the aspect ratio, you can proceed. If you don't then use the BASIC program ASPECT.BAS, included in your product diskette, to determine your aspect ratio.

The case *iopt* = 1 lets you handle distortions due to the first and second effects -- or at least provide a warning that distortion is required. This option computes the "proper" y value to assign to the upper row of the graph window. Remember that the x-axis scaling is set first so that *xmin* corresponds to *lpix* and *xmax* corresponds to *rpix*. Next, *ymin* is associated with *bpix*. The program (for *iopt* = 1) then computes the y-value corresponding to *rpix* taking into account the following two factors: (1) the ratio of y-axis units to x-axis units (*yoverx*) and (2) the screen aspect ratio (*aspect*). The following example provides details on how the revised value of *ymax* is computed. Understanding

these details is helpfull but not necessary, since the graphics subsystem carries out all the calculations for the user and supples the correct *ymax* for the user's choice of plot window size.

Suppose we want our units so that the x-axis represents years and the y-axis represents dollars. Our time span is 10 years and the dollar values range from zero to 5000 unit changes in the y direction. Unfortunately, this still doesn't guarantee the plot will fit into a square viewing region. The maximum y value for a given window, *yoverx*, *xmin*, *xmax*, aspect ratio and resolution is:

$$ymax = ymin + r1*r2*(xmax-xmin)$$

where $r1 = (tpix-bpix+1)/(rpix-lpix+1)$

and $r2 = (yoverx/aspect)*(8*\mathbf{ncol}/200)$.

Here **ncol** is 40 in the medium resolution and 80 in the high resolution mode. This is the expression computed when *iopt* = 1. (**ncol** is determined internally by reading the current mode). In our example, if we had a square viewing window (r1 = 1.0), an aspect ratio of 3/2 (a common value), *yoverx* = 500, and were in medium resolution, then the largest y value without distortion would be 5333 and the plot fits the window without distortion. When *iopt* = 1, 5333 is the value of *ymax* used in producing the graph. The scaling factors are computed using this computed value of *ymax*.

The mode values must be set by a call to @MODES prior to calling @WINDOW, otherwise the default values of graphics mode and text mode are used.

@FRAME - Definition of print page frame for graph

Syntax:

@FRAME(*xoff*, *yoff*, *xlen*, *ylen*)

Symbols:

xoff	: real*8	= x-offset in inches from top edge (default = 1.0)
yoff	: real*8	= y-offset in inches from left edge (default = 0.75)
xlen	: real*8	= length in x-direction in inches (default = 9.0)
ylen	: real*8	= length in y-direction in inches (default = 6.75)

This utility is used to specify a graph frame for printing, corresponding to the CRT window. The x-axis runs along the paper roll direction. The position *xoff*, *yoff* corresponds to the values *xmin*, *ymin* in user coordinates.

@XAXIS - Definition of x-axis

Syntax:

@XAXIS(*name, xst, xin, xmajor, minor, label, ndec*)

Symbols:

name	: character string	= name of the graph
xst	: real*8	= value of x at the start of the x-axis
xfin	: real*8	= value of x at the end of the x-axis
xmajor	: real*8	= distance between major (large) tic marks on the x-axis
minor	: integer	= flag for use of minor tic marks (0, + 1,-1)
label	: integer	= flag for auto-labeling of tic marks (0, + 1)
ndec	: integer	= number of decimal places for auto-labeling (0,1,2,3)

This utility generates the x-axis element for the named graph. The @WINDOW and/or @FRAME utilities must be called before @XAXIS is called.

The major tic marks along the x-axis will be placed at intervals of *xmajor* starting at *xorg*. If *xmajor* is positive/negative, then the tic marks will be placed above/below the x-axis respectively. If *xmajor* is zero, then no tic marks are inserted.

If *minor* = 0, then no minor tic marks are inserted. If minor = + 1, then minor tic marks are inserted halfway between the major tic marks (but are not labeled). If *minor* = -1, then a logarithmic set of tic marks are inserted (for log plots).

If *label* = 0, then there is no auto-labeling. If *label* = 1, the major tic marks are automatically supplied a label. If *label* = -1, then the origin is labeled as well. The format of the label is F10.n with the decimal point lined up with the major tic mark. The label is on the opposite side of the x-axis from the tic marks. Here n = *ndec* = number of decimal points desired. Note: If the absolute value of x exceeds 99999.999 at the tic mark then the label becomes 99999.999, except when *minor* = -1 (log axis), then the tic mark labels run from 10.***xst* ot 10.***xfin* rather that from *xst* to *xfin*.

For a log axis, the values of *xst* and *xfin* should be one of the following: 0.001, 0.01, 1.0, 10.0, 100.0, 1000.0 or 10000.0 with *xst* less than *xfin*.

@YAXIS - Definition of y-axis

Syntax:

@YAXIS(*name, yst, yin, ymajor, minor, label, ndec*)

Symbols:

name	: character string	= name of the graph
yst	: real*8	= value of y at the start of the y-axis
yfin	: real*8	= value of y at the end of the y-axis
ymajor	: real*8	= distance betrween major (large) tic marks on the y-axis
minor	: integer	= flag for use of minor tic marks (0, + 1,-1)
label	: integer	= flag for auto-labeling of tic marks (0, + 1)
ndec	: integer	= number of decimal places for auto-labeling (0,1,2,3)

This utility generates the y-axis element for the named graph. The @WINDOW and/or @FRAME utilities must be called before @YAXIS is called.

The major tic marks along the y-axis will be palced at intervals of *ymajor* starting at *yorg*. If *ymajor* is positive/negative, then the tic marks will be placed to the right/left of the y-axis respectively. If *ymajor* is zero, then no tic marks are inserted.

If *minor* = 0, then no minor tic marks are inserted. If *minor* = + 1, then minor tic marks are inserted halfway betreen the major tic marks (but are not labeled). If *minor* = -1, then a logarithmic set of tic marks are inserted (for log plots).

If *label* = 0, the there is no auto-labeling. If *label* = 1, the major tic marks are automatically supplied a label. If *label* = -1, then the origin is labeled as well. The format of the label is F10.n with the decimal point lined up with the major tic mark. The label is on the opposite side of the y-axis from the tic marks. Here n = *ndec* = number of decimal points desired. Note: If the absolute value of y exceeds 99999.999 at the tic mark then the label becomes 99999.999, except when *minor* = -1 (log axis), then the tic mark labels run from 10.***yst* ot 10.***yfin* rather that from *yst* to *yfin*.

For a log axis, the values of *yst* and *yfin* should be one of the following:

0.001, 0.01, 1.0, 10.0, 100.0, 1000.0 or 10000.0

with *yst* less than *yfin*.

@LABEL - Definition and positioning of labels in graph mode

Syntax:

@LABEL(*name, nchar, label, kolor, rpix, cpix*)

Symbols:

name	: character string	= name of the graph
nchar	: integer	= number of characters in label
label	: character string	= label to appear in graph
kolor	: integer	= color of label
rpix	: integer	= pixel row for start of label
cpix	: integer	= pixel column for start of label

This utility specifies a horizontal label or title to be written in the graph.

@TEXT - Definition and positioning of labels in text mode

Syntax:

@TEXT(*name, nchar, label, kolor, irow, icol*)

Symbols:

name	: character string	= name of the graph
nchar	: integer	= number of characters in label
label	: character string	= label to appear in graph
kolor	: integer	= color of label
irow	: integer	= text row for start of label
icol	: integer	= text column for start of label

This utility specifies a horizontal label or title to be written in the graph.

@CHRSIZE - Print character size definition

Syntax:

@CHRSIZE(size)

Symbol:

size	: real*8	= size in inches

This utility sets the character size for printed graphs. It has global effect in that the character size remains in effect until changed by a subsequent call. In the absense of this call, the default character size is 0.1 inches.

@NUMBER - Definition and positioning of numerical values

Syntax:

@NUMBER(*name, cpix, rpix, value, ndec*)

Symbols:

name	: character string	= name of graph
cpix	: integer	= starting pixel column
rpix	: integer	= starting pixel row
value	: real*8	= value of number
ndec	: integer	= number of decimal places (0-3)

This utility places a numerical value in the graph at position (cpix, rpix).

@XCLABEL - Definition of centered label for x-axis

Syntax:

@XCLABEL(*name, nchar, label, kolor*)

Symbols:

name	: character string	= name of graph
nchar	: integer	= number of characters in label
label	: character string	= text of label
kolor	: integer	= color index of label

This utility places text as a centered label below the x-axis.

@XELABEL - Definition of end label for x-axis

Syntax:

@XELABEL(*name, nchar, label, kolor*)

Symbols:

name	: character string	= name of graph
nchar	: integer	= number of characters in label
label	: character string	= text of label
kolor	: integer	= color index of label

This utility places text as an end label below the x-axis.

@YCLABEL - Definition of centered label for y-axis

Syntax:

@YCLABEL(*name, nchar, label, kolor*)

Symbols:

name	: character string	= name of graph
nchar	: integer	= number of characters in label
label	: character string	= text of label
kolor	: integer	= color index of label

This utility places text as a centered label to the left of the y-axis.

@YELABEL - Definition of end label for y-axis

Syntax:

@YELABEL(*name, nchar, label, kolor*)

Symbols:

name	: character string	= name of graph
nchar	: integer	= number of characters in label
label	: character string	= text of label
kolor	: integer	= color index of label

This utility places text as an end label to the left of the y-axis.

@SETUP - Properties of curves and point sets

Syntax:

@SETUP(*name, ident, ndots, kolor, symbol, klrsym*)

Symbols:

name	: character*8	=	name of graph
ident	: character*2	=	identifier of function
ndots	: integer	=	number of pixels (dots) in a dash
kolor	: integer	=	color index of curve(s) in function
symbol	: integer	=	index of ASCII symbol or negative flag
klrsym	: integer	=	color index of the symbol

This utility sets up the properties of graph functions to be generated by the @CURVE or @POINT utilities. Each function in the graph must have a unique 2-character identifier ident.

If *ndots* = 0 then a solid line is drawn. If *ndots* = 1 then no line is drawn. For *ndots* greater than 1, dashed lines are drawn with the dashes having *ndots* pixels. The color of the solid or dashed line is specified by the *kolor* index.

If *symbol* = -1, then the default (cross) is drawn at the data points generated by @CURVE or @POINT. If *symbol* = -2 then no symbol is drawn. Otherwise the character with ASCII index symbol is drawn. If a symbol is drawn, then its color is given by the *klrsym* index.

@CURVE - Connected points on a curve function

Syntax:

@CURVE(*name, ident, xpoint, ypoint*)

Symbols:

name	: character*8	= name of graph
ident	: character*2	= identifier of curve function
xpoint	: real*8	= abscissa value of a point
ypoint	: real*8	= ordinate value of a point

This utility is called repeatedly to enter points of a function into the graph object. These points are tabulated sequentially in a table file of the object identified by *ident*. When the graph is finally produced from the object (via @SHOW or @DRAW), the resulting table (all of the points together) will be plotted in the order in which the first point is tabulated in the sequence of the elements of the graph. Thus, although the tabulation of points is intermittant, and may be interleaved with the tabulation of the points of other functions, each table is separate, and when the graph is produced, each table is plotted contiguously in the order in which the first point of each table was tabulated in the graph object. In this case, the points are joined by a straight line having the dashed properties, color, and symbol annotation defined for the corresponding *ident* by @SETUP.

@SPLINE - Cubic spline interpolation of a curve function

Syntax:

@SPLINE(*name, ident, xpoint, ypoint*)

Symbols:

name	: character*8	= name of graph
ident	: character*2	= identifier of curve function
xpoint	: real*8	= abscissa value of a point
ypoint	: real*8	= ordinate value of a point

This utility is called repeatedly to enter points of a function into the graph object. These points are tabulated sequentially in a table file of the object identified by ident. When the graph is finally produced from the object (via @SHOW or @DRAW), the resulting table (all of the points together) will be plotted in the order in which the first point is tabulated in the sequence of the elements of the graph. Thus, although the tabulation of points is intermittant, and may be interleaved with the tabulation of the points of other functions, each table is separate, and when the graph is produced, each table is plotted contiguously in the order in which the first point of each table was tabulated in the graph object. In this case the resulting curve in between the tabulated points is filled in by interpolating a cubic spline fit of the tabulated points, resulting in a smooth curve that passes through all of the points. The curve will have the dashed properties, color, and symbol annotation defined for the corresponding ident by @SETUP.

@POINT - Discrete (unconnected) points on a curve function

Syntax:

@POINT(*name, ident, xpoint, ypoint*)

Symbols:

name	: character*8	= name of graph
ident	: character*2	= identifier of curve function
xpoint	: real*8	= abscissa value of a point
ypoint	: real*8	= ordinate value of a point

This utility is called repeatedly to enter points of a function into the graph object. These points are tabulated sequentially in a table file of the object identified by ident. When the graph is finally produced from the object (via @SHOW or @DRAW), the resulting table (all of the points together) will be plotted in the order in which the first point is tabulated in the sequence of the elements of the graph. Thus, although the tabulation of points is intermittant, and may be interleaved with the tabulation of the points of other functions, each table is separate, and when the graph is produced, each table is

plotted contiguously in the order in which the first point of each table was tabulated in the graph object. In this case only the points are plotted, and symbols must have been selected via @SETUP for these points to appear.

@CONTDEF - Definition of contour parameters for set of contours

Syntax:

@CONTDEF(*name, ident, lcolor, ucolor, mcolor, legend*)

Symbols:

name	: character*8	=	name of graph
ident	: character*2	=	identifier of contour function
lcolor	: integer	=	color index of labeled contours
ucolor	: integer	=	color index of unlabeled contours
mcolor	: integer	=	color index of marker labels on labeled contours
legend	: integer	=	legend flag

This utility performs the setup function for a contour function. It must be called prior to @CONTOUR. If *legend* = 0, then no legend is produced by @CONTOUR, otherwise a legend will be produced.

@MESH - Contour function mesh generation

Syntax:

@MESH(*name, ident, xmin, xmax, ymin, ymax, nx, ny, cfun*)

Symbols:

name	: character*8	=	name of graph
ident	: character*2	=	identifier of contour function
xmin	: real*8	=	minimum value of x-coordinate
xmax	: real*8	=	maximum value of x-coordinate
ymin	: real*8	=	minimum value of y-coordinate
ymax	: real*8	=	maximum value of y-coordinate
nx	: integer	=	mesh size in x-direction
ny	: integer	=	mesh size in y-direction
cfun	: real*4	=	external FUNCTION subprogram defining contour function

This utility must be called prior to @CONTOUR. It generates a 2-dimensional table

of the contour function by calling the external function subprogram once for each mesh point in the table. This table is written to a file of the graph object which is subsequently used in producing the contour plot via one or more calls to @CONTOUR.

@CONTOUR - Generation of contour curves

Syntax:

@CONTOUR(*name, ident, xscale, values, lbflg, numcon, idef*)

Symbols:

name	: character*8	=	name of graph
ident	: character*2	=	identifier of contour function
xscale	: real*8	=	aspect ratio of CRT
values	: real*8	=	array of *numcon* contour values
lbflg	: integer	=	array of *numcon* contour flags
numcon	: integer	=	number of contour curves (1-10)
idef	: integer	=	plot size parameter

This utility generates as series of contours each corresponding to a fixed value of the contour function specified by *ident*. The *values* array contains the *numcon* values to be assigned to the contours. The *lbflg* array is a set of flags (1 or 0) indicating whether a given contour should be labeled (1) or unlabeled (0).

This utility is restricted to 10 contours per call. For more contours on a given function, successive calls can be made with a different *values* array and numcon. However, all labeled contours must be in the first call. In successive calls, the *lbflg* array should be set to zero. Also, *idef* must be set to 1 or 2 to avoid having the prior contours cleared during successive calls.

The *idef* parameter controls the use of default properties. If *idef* = 0, the default plot size, graphics mode, contour line colors and legend are used. If *idef* = 1, the plot size is set by default and the line colors and legend are defined by @CONTDEF, which must precede the call to @CONTOUR for the associated *ident*. If *idef* = 2, the line colors and legend are defined by the @CONTDEF, but the plot window must be defined by a preceding call to @WINDOW. In the latter cases (*idef* = 1,2), graphics mode must be set by @MODES before the graph is produced by @SHOW or @DRAW.

8.2. Graph Production Command Calls

@MODES - Definition of monitor graphics and text modes

Syntax:

@MODES(*graphics*, *text*)

Symbols:

graphics: integer - default = 18

text: integer - default = 3

This utility sets the graphics mode for the particular display monitor. The following modes are permitted on IBM-compatible display monitors:

MODE	TYPE	COLS	ROWS	COLORS	CHAR-SIZE	PAGES
0	text	40	25	2	8x8	0-7
1	text	40	25	16	8x8	0-7
2	text	80	25	2	8x8	0-7
3	text	80	25	16	8x8	0-7
4	graph	320	200	4	8x8	0-
5	graph	320	200	4	8x8	0-
6	graph	640	200	2	8x8	0-
7	text	80	25	2	9x14	0-7
8	graph	640	400	2	8x16	0
13 EGA	graph	320	200	16	8x8	0-7
14 EGA	graph	640	200	16	8x8	0-7
15 EGA	graph	640	350	4	8x14	0-1
16 EGA	graph	640	350	16	8x14	0-1
17 VGA	graph	640	480	2	8x16	0
18 VGA	graph	640	480	16	8x16	0
19 VGA	graph	320	200	256	8x8	0

This utility sets the display modes to conform to the users equipment. This command-call must precede first call to @SHOW if default values are inappropriate.

@SHOW - Graph display

Syntax:

@SHOW(*name*)

Symbols:

name: character*8 - name of graph

This utility causes the graph to be displayed on the console monitor. This display will remain on the monitor until the enter key is pressed, allowing the program to continue.

@DRAW - Write graph print file

Syntax:

@DRAW(*name*)

Symbols:

name : character*8 = name of graph

This utility writes the named graph to a print file of the same name with file extension ".GPR". This file is closed.

@CLEAR - Delete graph object from database

Syntax:

@CLEAR(*name*)

Symbol:

name : character*8 = name of graph

This utility removes the named graph object from the database to free space for additional objects. It is applied after a graph has been produced, deleting the synthesis files used in the production. Thus it does not affect output of graphs that have already been produced.

8.3. X-Y Plotting Applications

In view of the economy of problem expression provided by calculus-level macro statements, graphics commands for those programs employing graphics tend to dominate problem coding. Moreover, since this chapter on graphics follows all of the chapters dealing with the calculus operations, it is convenient to present applications problems here that utilize combinations of features, including graphics. In the following discussions, these applications are presented in their entirety and topically discussed with emphasis on usage of the graphics operations, but with also some discussion regarding the calculus operations used in the applications.

Applications Problem 8-1: Admittance Circuit Fit

This application was introduced as illustration problem 3 in Section 2.1, which gives a circuit diagram and a discussion of the model. The graphics-enhanced version of this program is given below.

In the problem procedure, two command-calls to the procedures AXES and FINISH serve to setup graph synthesis for the program and to produce the final graph image respectively.

```
      GRAPHICS BOTH
      PROBLEM CIRCUIT (5000,1000,1000)
        COMMON/PARAMS/EL,ELS,CS
        DIMENSION F(21),Y(21),R(21),W(21),YC(21)
        DATA Y/1.273,1.278,1.392,1.604,1.708,1.950,2.148,2.297,
     -         2.503,2.893,3.305,4.005,5.077,8.069,14.84,39.47,
     -         -15.06,-9.705,-7.368,-5.286,-3.907/
        EL=1.1E-10 : ELS=-1.1E-10 : CS=1.1E-12
        DO 10 I=1,21
          F(I)=2850+50*I
          W(I)=6.28318E6*F(I)
 10     CONTINUE
        @AXES('CIRFIT',F,Y,YC,W)
        FIND EL,ELS,CS; IN FIT(F,Y,R,W,YC);
     -    BY AJAX; TO MATCH R
        @FINISH('CIRFIT',F,YC,W)
      END

      MODEL FIT(F,Y,R,W,YC)
        COMMON/PARAMS/EL,ELS,CS
        DIMENSION F(*),Y(*),R(*),W(*),YC(*)
        DO 10 I=1,21
          YC(I)=-1/(W(I)*EL-1/(W(I)*CS-1/(W(I)*ELS)))
          R(I)=1/YC(I)-1/Y(I)
 10     CONTINUE
      END
```

```
      PROCEDURE AXES(GNAME,F,Y,YC,W)
        COMMON/PARAMS/EL,ELS,CS
        DIMENSION F(*),Y(*),YC(*),W(*)
        CHARACTER*(*) GNAME
        @GRAPH(GNAME,'2DGRAPH')
        @CHRSIZE(0.05)
        @FRAME(0,.5,5,4)
        @WINDOW(GNAME,50,500,30,400,29D2,39D2,-20,40,2900,0,0,1,1.5)
        @SETUP(GNAME,'Y ',0,0,ICHAR('*'),14) ! Measured (Red *'s)
        @SETUP(GNAME,'Y0',0,10,-2,0)
        @SETUP(GNAME,'YC',0,12,-2,0)
        @XAXIS(GNAME,2900,3900,200,0,1,1)
        @YAXIS(GNAME,-20,40,10,0,1,1)
        @XELABEL(GNAME,15,'FREQUENCY (CPS)',7)
        @YELABEL(GNAME,10,'ADMITTANCE',7)
        DO 10 I=1,21
          YC(I)=-1/(W(I)*EL-1/(W(I)*CS-1/(W(I)*ELS)))
          @POINT(GNAME,'Y ',F(I),Y(I))
          @CURVE(GNAME,'Y0',F(I),YC(I))
10      CONTINUE
        @LABEL(GNAME,13,'INITIAL GUESS',10,70,90,0)
        @LABEL(GNAME,9,'FINAL FIT',12,70,175,0)
      END

      PROCEDURE FINISH(GNAME,F,YC,W)
        COMMON/PARAMS/EL,ELS,CS
        DIMENSION F(*),YC(*),W(*)
        CHARACTER*(*) GNAME
        DO 10 I=1,21
          YC(I)=-1/(W(I)*EL-1/(W(I)*CS-1/(W(I)*ELS)))
          @CURVE(GNAME,'YC',F(I),YC(I))
10      CONTINUE
        @SHOW(GNAME)
      END
```

AXES Procedure - This procedure creates the graph object with the command @GRAPH. It establishes the absolute character size at 0.2 inches for printing on a 300 DPI laser printer, and defines a 5 inch by 4 inch frame in the lower-left corner of the printer's raster (in order to make the graphics capture file small enough to easily import into Ventura publisher). The plotting window of the graph is then generated via the @WINDOW command. This establishes the scaling for the subsequent functions to be plotted.

The subsequent @SETUP commands create the identity of the three functions to be plotted:

- 'Y ' - The discrete data points of the measurements represented by yellow asterisks;
- 'Y0' - The green curve of the admittance function based on the initial guess of its parameters; and
- 'YC' - The red curve of the admittance function based on the final convergence values of the parameters.

These identifiers are later used as sort keys to sort the data entered into the graph object

via @POINT and @CURVE commands, so that the separate functions are accumulated individually even though the functions are generated concurrently.

The subsequent commands @XAXIS, @YAXIS, @XELABEL, and @YELABEL establish and label the two coordinate axes.

Next, the two initially known functions (data points and the admittance function based upon guessed parameters) are generated and entered into the database.

Finally, labels are placed in the graph in proximity to the functions they describe.

FINISH procedure - The FINISH procedure is used to generate the 'YC' (converged) version of the admittance function in the graph to show goodness of fit. The DO loop computes the YC function and uses @CURVE enter it into the graph object of the database.

Graph Production - Up to this point all graphics operations have been part of the synthesis phase which accumulates the graph object in the database. The production of a graphical image from this object is accomplished via the @SHOW command (for screen display) or the @DRAW command (for print file generation). However, the particular form of graphics production is also governed by the GRAPHICS metacommand which precedes the PROBLEM statement. When this command is present, it overrides the type of the production command, causing output to be routed to the "SCREEN", "FILE", or "BOTH". In this application, the graph image was routed to "BOTH" screen and print file. The resulting screen and print images are shown in Figures 8-1 and 8-2.

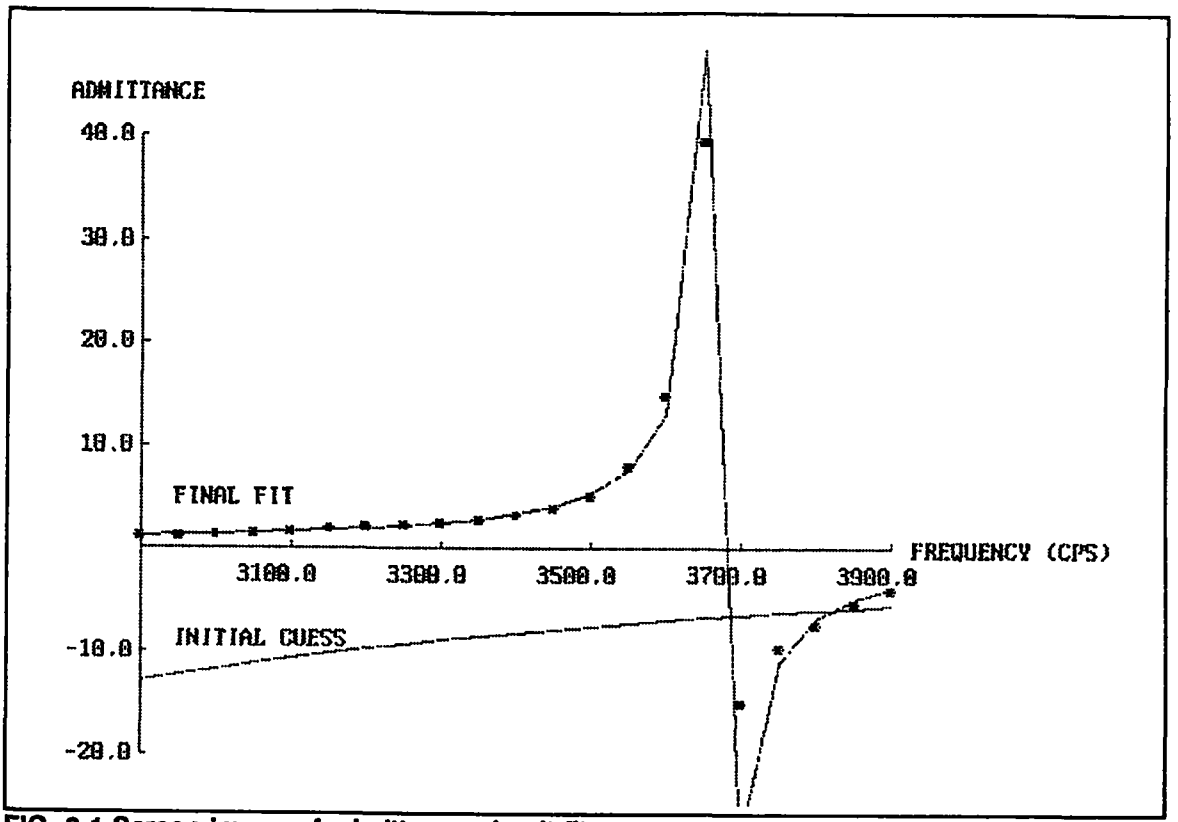

FIG. 8-1 Screen image of admittance circuit fit

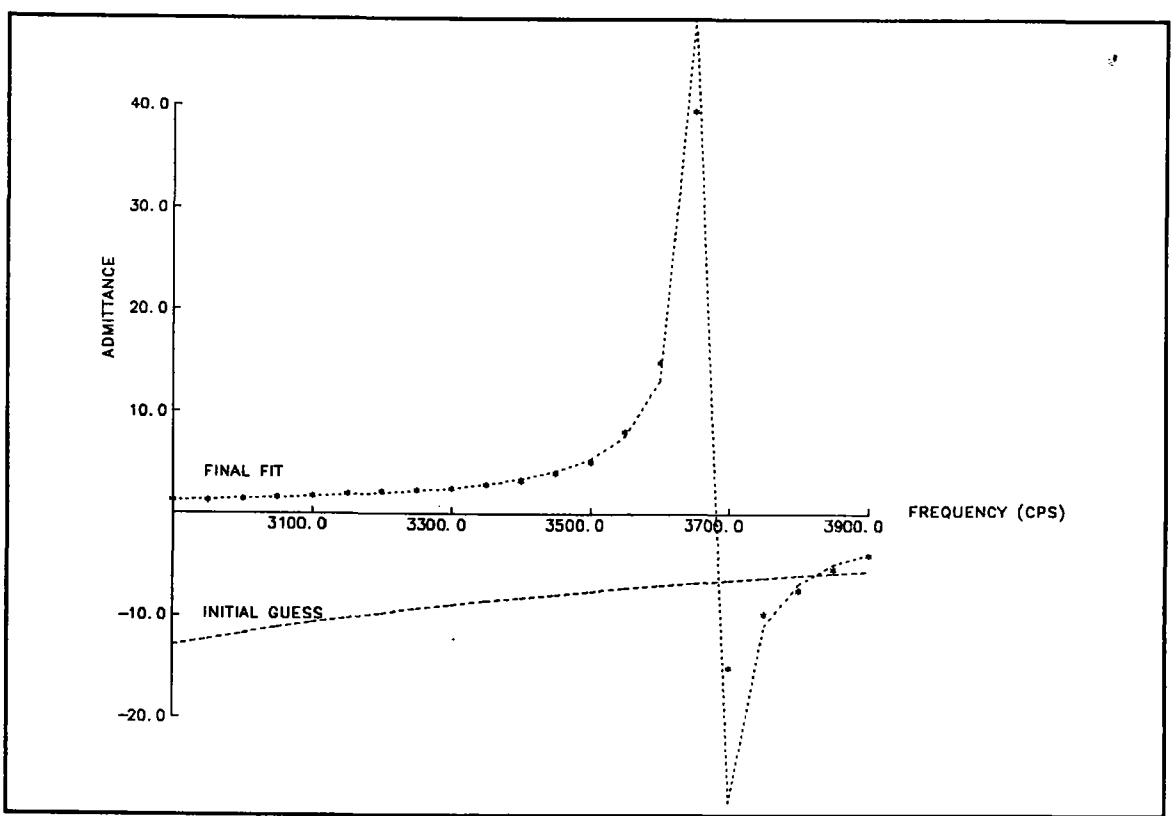

FIG. 8-2 File image of admittance circuit fit

Application Problem 8-2: Pilot Ejection Profile

This problem has been used for many years as an example to demonstrate continuous simulation languages using a trial & error trajectory simulation to characterize safe ejection in a piecemeal fashion. This version is somewhat different, in that the whole problem of profiling the regime of safe ejection is completed in one run using implicit equations solver AJAX to solve a series of two-point boundary value problems.

```
      GLOBAL ALL
      GRAPHICS FILE
      PROBLEM EJECT  ! Pilot Ejection Profile
        DIMENSION ALT(6),VEL(6)
        CHARACTER FCSINT*2,PLANE*7
        DATA ALT/0,10000,20000,30000,40000,50000/
        SMASS=7 : G=32.2 : CD=1 : VE=40 : THETAD=15 : TIME=1.0
        G=10 : Y1=4 : VA=100
        DO 10 I=1,6
          H=ALT(I) : IT=0 : PLANE='PLANE'//FCSINT(I)
          @AXES(PLANE,'PILOT EJECTION')
          IF(H.LE.35332) THEN
             RHO=0.002378*(1-.689E-5*H)**4.256
          ELSE
             RHO=0.00315/EXP(1.452+(H-35332)/20950)
          ENDIF
          FIND VA,TIME; IN SEAT(PLANE); BY AJAX(ACON); TO MATCH GX,GY
          @DISPLAY(PLANE)
          VEL(I)=VA
   10   CONTINUE
        @SAFE('PROFILE')
      END

      MODEL SEAT(PLANE)
        CHARACTER*2 FCSINT,NI,PLANE*7
        VX=SQRT(VA**2)-VE*SIND(THETAD) : VY=VE*COSD(THETAD)
        V=SQRT(VX*VX+VY*VY) : THETA=ATAN(VY/VX)
        X=0 : Y=Y1 : T=0 : DT=ABS(TIME)/20 : DP=4*DT : TP=T+DP
        IT=IT+1 : NI=FCSINT(IT)
        IF(IT.GT.9) THEN
          @POINT(PLANE,'I'//NI,X,Y)
        ELSE
          @CURVE(PLANE,'I'//NI,X,Y)
        ENDIF
        INITIATE ISIS; FOR MOTION; EQUATIONS
     &     THEDOT/THETA, VDOT/V, XDOT/X, YDOT/Y; OF T; STEP DT; TO TP
        DO WHILE (T.LT.TIME)
          INTEGRATE MOTION; BY ISIS
          @CURVE(PLANE,'I'//NI,X,Y)
          TP=TP+DP
        END DO
        GX=-X-30       ! Boundary condition on X at T=TIME
        GY=Y-20        ! Boundary condition on Y at T=TIME
        TERMINATE MOTION
      END

      MODEL MOTION    ! Differential Equations
        D=0.5*RHO*CD*S*V*V
        THEDOT=-G*COS(THETA)/V
        VDOT=-D/SMASS-G*SIN(THETA)
        XDOT=V*COS(THETA)-ABS(VA)
        YDOT=V*SIN(THETA)
      END
```