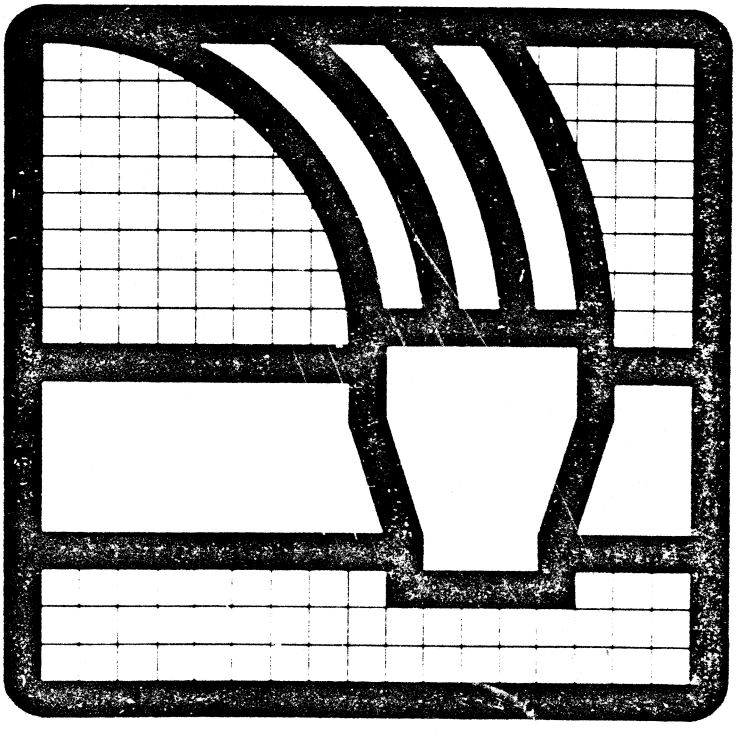
CalComp

International

Graphics



Part No. 10255-901-017-1

CalComp

Model 81 Plotter

Host Computer Basic Software

Manual

August 1981

CALCOMP
INTERNATIONAL DIVISION
SANDERS

Printed in the United States of America

Continuing research and improvement may result in specification changes at any time.

CONTENTS

ILLUSTRATIONS

TABLES

1. INTRODUCTION

This manual provides an introductory technical description of CalComp's Basic Software Package for the Model 81 Plotter. It includes the information that a FORTRAN oriented programmer needs to write graphic computer programs.

1.1 TYPES OF SOFTWARE

CalComp provides graphic software in three general categories:

- APPLICATIONS PROGRAMS
- FUNCTIONAL SOFTWARE
- BASIC SOFTWARE

The highest level of graphic software is the Applications Program. An Application Program is a complete problem solver. A user need only supply data and select among program options to obtain the desired graphical output - no programming is required on the user's part. Applications Programs are written in a higher level language, usually FORTRAN IV, and are available for a variety of computers.

Typical CalComp applications programs are GPCP (General Purpose Contouring Program) and Datagraph (Produces Management information charts and graphs directly from user data on cards, tape or disc.)

Appendix B contains a short description of the applications software packages currently available from CalComp. The Applications Program normally uses a Basic Software Package and may also make use of CalComp Functional Software.

Functional Software is an intermediate level of software which relieves the user of programming many commonly used graphic functions. Host computer resident functional software is usually written in FORTRAN and grouped into packages by general application:

- BUSINESS
- DRAFTING
- SCIENTIFIC
- GENERAL

Each package consists of a set of subroutines to perform particular operations. Appendix C contains a list of current Functional Software.

The lowest level of software used in producing a plot is termed Basic Software. Host Computer Basic Software (HCBS) consists of a set of subroutines which allow the programmer to perform elementary plotting operations such as drawing lines or annotation, selecting a pen, scaling the plot, etc. The basic software generates and formats the command codes necessary to perform the specified operation and transmits them to the plotter in a highly compact and efficient format. These codes are then interpreted by the plotter firmware and expanded into commands necessary to control the plotter. The software/hardware interrelationships are shown schematically in Figure 1-1.

This manual provides an explanation of the subroutines included in the Model 81 Host Computer Basic Software package and their use to produce graphical output.

1.2 SOFTWARE SUPPORT POLICY

Software is an integral and essential part of the CalComp product line. Like individual hardware items, each software package is given a product number and is supported with appropriate literature and documentation. Because of the proprietary nature of CalComp software, the packages are leased rather than sold outright. The lease agreement allows unrestricted use of the software with your CalComp system and the lease price covers this usage for as long as you retain the system.

CalComp personnel are available for consulting and assistance to users of the software. Contact your local office for instructions on obtaining assistance.

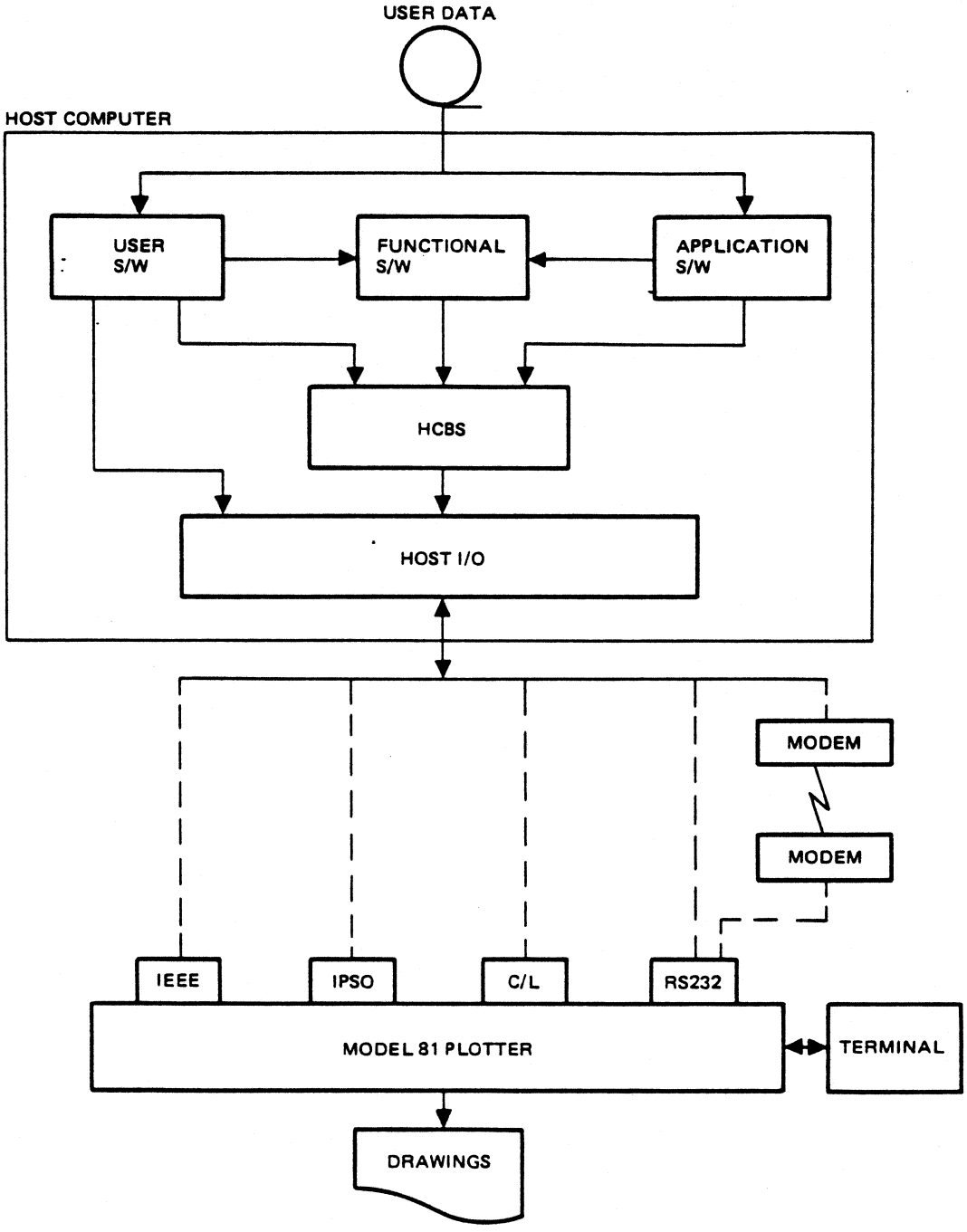


Figure 1-1 Hardware/Software Configurations

1.3 STANDARD SOFTWARE MAKES IT EASY

Section two describes in detail CalComp's Host Computer Basic Software (HCBS). This software package consists of a set of subroutines written in FORTRAN which control elementary operations of the plotter and provide certain aids in plotting graphs. These subroutines are called by user written routines, applications programs and host computer functional software.

The HCBS subroutines are callable directly from FORTRAN programs, and, by observing some special considerations, may also be called from assembly language, PL/I, or COBOL programs.

Standard FORTRAN name conventions are used throughout this manual and in the HCBS package.

The subroutines included as part of the standard HCBS package are as follows:

PLOT
Plots straight line between two points; establishes plot origin

PLOTS -
Initializes system

FACTOR -
Scales entire plot

WHERE -
Returns current pen location

NEWPEN -
Selects pen

SYMBOL -
Plots annotation and special symbols

NUMBER -
Plots the decimal equivalent of an internal floating point number

SCALE -
Determines starting value and scale for an array of data to be plotted on a graph

AXIS -
Draws an annotated axis line for a graph

LINE -
Scales and plots a set of data points defined by an X and Y coordinate array

CIRCLE -
Plots circles and circular arcs

DASHS -
Enables dashline plotting

WINDOW -
Defines inclusive plot window

CHTADV -
Advances paper with optional programmable paper advance

The subroutines PLOT, SYMBOL, NUMBER, SCALE, AXIS, LINE are basic routines compatible as far as possible with other CalComp plotters and software.

CIRCLE and DASHS are upwards compatible with the CalComp 907 based family of Plotters.

WINDOW and CHTADV are specific to the features of the Model 81.

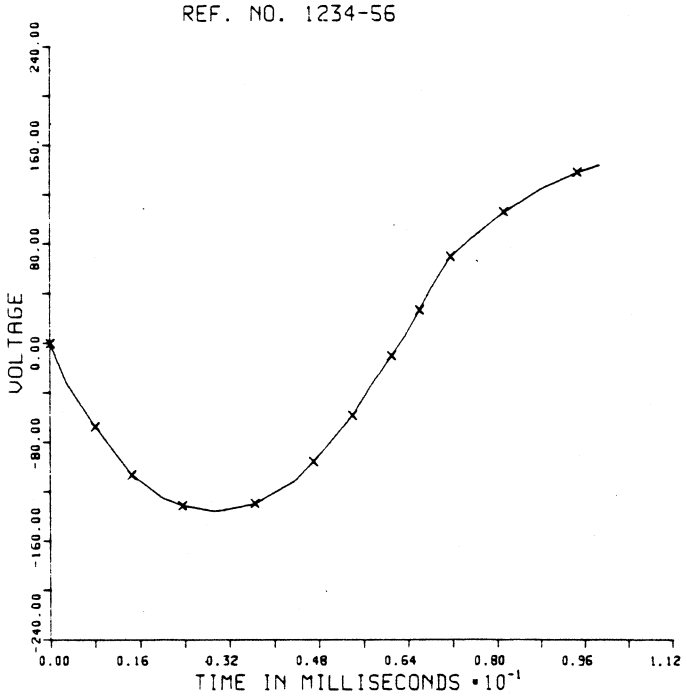
Appendix A contains a summary of the standard calling sequences for these routines.

1.3.1 A SAMPLE PLOTTING PROGRAM

To illustrate the use of CalComp Host Computer Basic Software, a sample program is given which will produce the graph shown in Figure 1-2. The only assumption made is that:

The 24 pairs of TIME and VOLTAGE data values are contained in a file of 24 records;

Notice that only 11 executable statements are required to complete the graph.



INPUT DATA

X	Y
0.0	0.0
0.0030	-33.0
0.0080	-67.0
0.0110	-86.0
0.0145	-106.0
0.0200	-125.0
0.0235	-131.0
0.029	-136.0
0.0365	-129.0
0.0435	-111.0
0.047	-95.0
0.0495	-82.0
0.0540	-58.0
0.0570	-36.0
0.0610	-10.0
0.0635	7.0
0.0660	27.0
0.0680	44.0
0.0715	70.0
0.0750	84.0
0.0810	106.0
0.0875	124.0
0.0945	138.0
0.0985	144.0

Figure 1-2. Sample Graph

PERFORMANCE TEST FORTRAN PROGRAM

```
      DIMENSION XARRAY (26), YARRAY (26)
```
Reserve space for 24 data values plus two additional locations required by the SCALE, AXIS, and LINE subroutines.

```
10    CALL PLOTS (0, 0, 6)
```
Initialize the PLOT subroutine with the logical number of the output device.

```
20    READ 25, (XARRAY (I), YARRAY (I), I = 1, 24)
```

```
25    FORMAT (2F5.4)
```
Read 24 pairs of TIME and VOLTAGE from an input file into two arrays with names XARRAY and YARRAY.

```
30    CALL PLOT (2.0, 2.0, -3)
```
Establish a new origin two centimeters higher and to the right of the point where the pen was initially placed by the operator so that the annotation of the axes will fit between the axis and the edge of the plotting surface for each axis.

```
40    CALL SCALE (XARRAY, 14.0, 24, 1)
```
Compute scale factors for use in plotting the TIME values within a fourteen centimeter plotting area.

```
50    CALL SCALE (YARRAY, 12.0, 24, 1)
```
Compute scale factors for use in plotting the VOLTAGE data values within a 12 centimeter plotting area (i.e. the data pairs of TIME, VOLTAGE will plot within a ten-by-twelve cm area).

```
60    CALL AXIS (0.0, 0.0, 20HTIME IN MILLISECONDS, -20, 14.0, 0.0, XARRAY (25),
      XARRAY (26))
```
Draw the TIME axis (14 cms long), using the scale factors computed in statement 40 to determine the milliseconds at each cm along the TIME axis.

```
70    CALL AXIS (0.0, 0.0, 7HVOLTAGE, 7, 12.0, 90.0, YARRAY (25), YARRAY (26))
```
Draw the VOLTAGE axis (12 cms long), using the scale factors computed in statement 50 to determine the voltage at each cm along the VOLTAGE axis.

```
80    CALL LINE (XARRAY, YARRAY, 24, 1, 2, 4)
```
Plot VOLTAGE vs TIME, drawing a line between each of the 24 scaled points and a symbol X at every other point.

```
90    CALL SYMBOL (2.0, 14.0, 0.70, 16HPERFORMANCE TEST, 0.0, 16)
```
Plot the first line of the graph title.

```
100   CALL SYMBOL (3.0, 12.5, 0.50,16HREF. NO. 1234-56, 0.0, 16)
```
Plot the second line of the graph title.

```
110   CALL PLOT (12.0, 0.0, 999)
```
Advance the pen beyond the current plotting area, advance the paper (optional), write a terminating record, and close the plot output device.

```
120   STOP
```
Terminate program execution.

```
      END
```

1.3.2 PLANNING YOUR GRAPH

Graphs and plots, like computer reports, require some planning to achieve a pleasing and effective format. The following checklist of plotting conventions is letter-keyed to Figure 1-3 to help you in your planning.

(A) When the plotter is turned on or the software is initialized, the pen is positioned at the physical origin which coincides with the current logical origin. All pen movements are defined in X and Y coordinates relative to the current logical origin. Subsequent origins can be established at other positions by appropriate programming (see PLOT).

If more than one plot is to be drawn, an origin for each should be established, with care taken that the plots do not overlap.

(B) The X-axis lies parallel to the front of the plotter, with the +X direction to the right.

The maximum width of a plot in the X-direction is the length of the platen (33.8 cms). It is possible to produce several full size plots without operator intervention by the use of the optional roll paper advance hardware.

(C) The Y axis of the plotter is parallel to the beam which carries the pen. The maximum height of a plot is 28.0 cms.

(D) Angles are measured in degrees from the horizontal (parallel to the X axis) extending to the right. Positive angles are drawn counterclockwise, negative angles are drawn clockwise. Any angle argument used in a calling sequence may be stated in plus or minus degrees relative to the X-axis.

(E) When drawing several plots in one program, it may be desirable to draw trim lines for the operator's convenience in separating the plots.

(F) After the last plot has been drawn, the pen should be moved to a position that permits easy removal of all plots. e.g. coordinate 0,0.

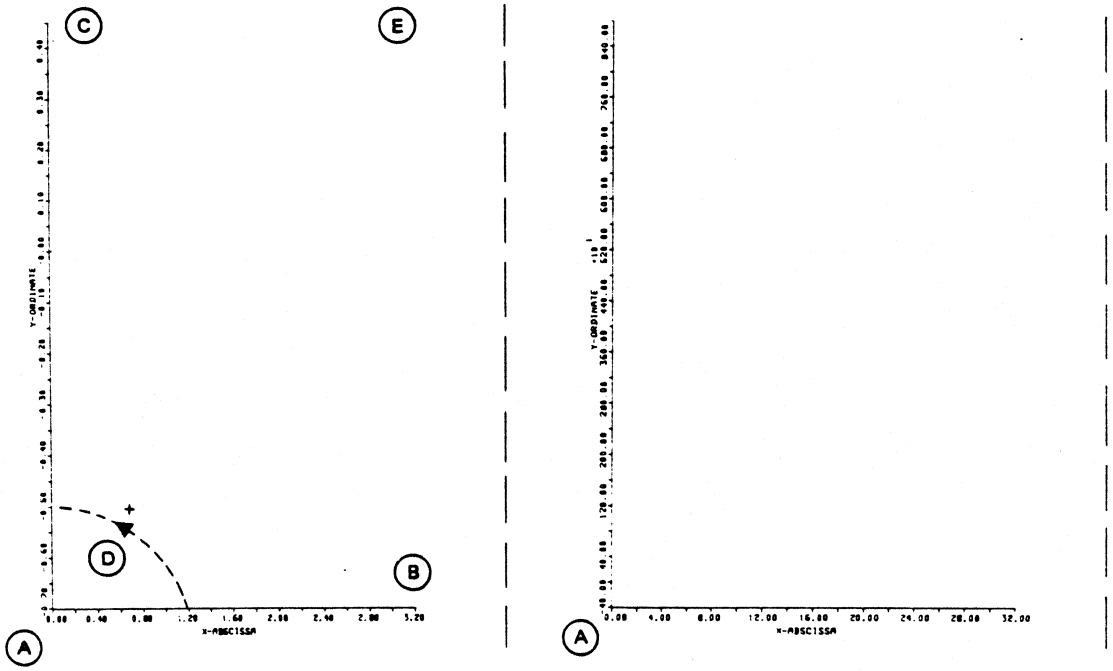


Figure 1-3. Plotting Conventions

2. GRAPHIC SUBROUTINES

Many graphic applications require the generation of X-Y graphs to show the relationship between two or more sets of data. Usually these graphs can be produced easily and quickly by a suitably programmed combination of the five basic supporting subroutines SCALE, AXIS, LINE, SYMBOL and NUMBER. These subroutines do not directly produce plotter commands; they only compute appropriate arguments that define pen positions, and then call the PLOT subroutine which generates the actual plotter commands. The subroutines DASHS and CIRCLE fulfill additional common graphic requirements. The CHTADV and WINDOW routines provide facilities to help the user organize a plot job.

When plotting requirements cannot be satisfied by using the supporting subroutines, the user can call the PLOT subroutine, which gives him direct control of pen movement (to any X, Y coordinate position), pen status (up or down), and definition of a logical origin.

Four other functions are closely associated with the PLOT operation as follows:

PLOTS
Performs initialization.

FACTOR
Adjusts the overall size of a plot.

WHERE
Returns the current pen location.

NEWPEN
Selects pens.

Each of these subroutines is described in the following paragraphs.

2.1 PLOTS SUBROUTINE

The PLOTS subroutine is used to initialize the HCBS. It must be called only once before any other call to an HCBS routine is made. This call sets up certain constants and opens the plot output device by performing standard opening procedures through the computer's operating system, and in some configurations initializes specific I/O parameters in the 81 plotter. PEN 1 is selected and moved to the plotter origin at the bottom left hand corner of the plotting surface. Figure 2-1 includes an example of the use of PLOTS.

The calling sequence has three arguments:

CALL PLOTS (0, 0, LDEV)

The first two arguments of the PLOTS subroutine are no longer used, but are maintained for compatibility with versions of the basic software package for other CalComp products.

LDEV
is the logical output device number, i.e., the device number of the plotter.

2.2 PLOT SUBROUTINE

The PLOT subroutine is used to move the pen in a straight line to a new position, with the pen either up or down during the movement. It converts the arguments to the appropriate sequence of plotter commands and outputs these to the plotter.

The calling sequence has three arguments:

CALL PLOT (XPAGE, YPAGE, ± IPEN)

XPAGE, YPAGE
are the X,Y coordinates, in cms to which the pen is to be moved. The pen is always moved from the current position to the new coordinates in a straight line. An origin (where both X, Y equal zero) may be established anywhere on (or off) the plotting surface, as explained below for negative IPEN values.

±IPEN
is a signed integer which controls pen status (up or down) and origin definition.

If IPEN = 2, the pen is down during movement, thus drawing a visible line.

If IPEN = 3, the pen is up during movement.

If IPEN = -2 or -3, a new origin is defined at the new position as if IPEN were positive. The logical X, Y coordinates of the new pen position are set equal to zero. This position is the reference point for succeeding pen movements. At this time all of the plotter commands accumulated in the output buffer are transmitted to the output device.

IPEN = 999 must be used to signify the end of a Plot Job and must be the last call to the HCBS. The effects are the same as for IPEN = -3, with the addition that if the roll paper advance is attached, the paper will be advanced XPAGE cms scaled by the current FACTOR. The output device is also closed.

The examples in Figure 2-1 and 2-2 show the pen movements that result from a series of calls to the PLOT subroutine. The initial call to PLOTS, as well as a call to FACTOR, are included.

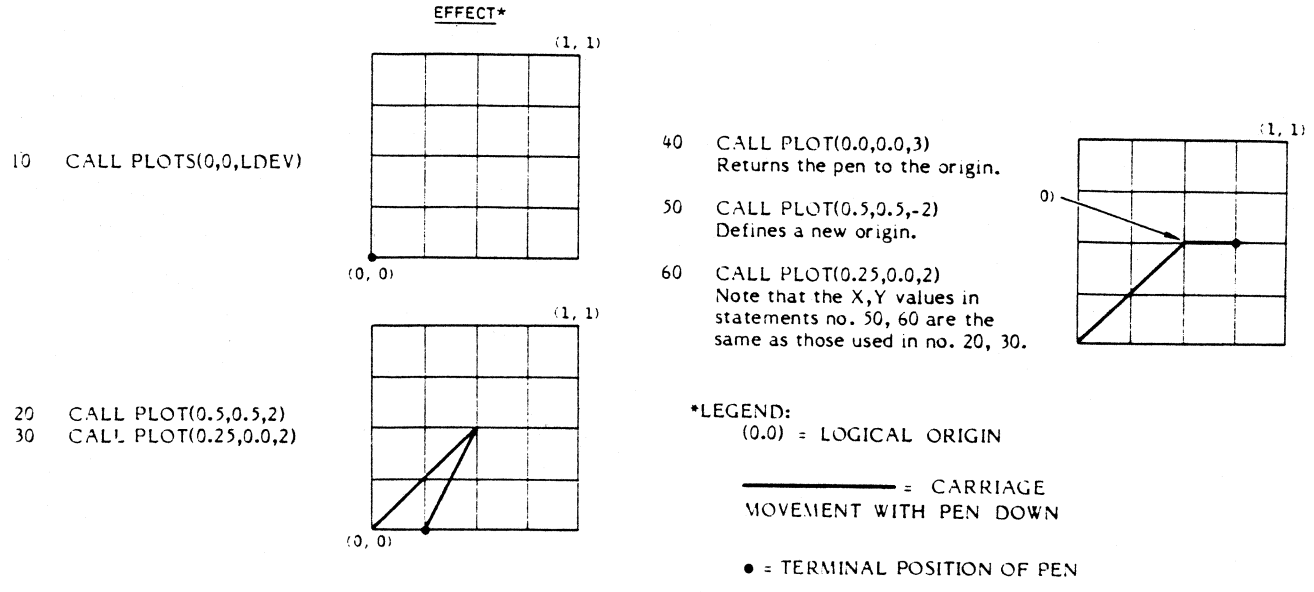


Figure 2-1. Examples of PLOT and PLOTS

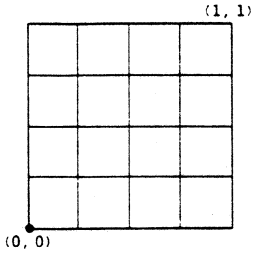
2.3 FACTOR SUBROUTINE

The FACTOR subroutine enables the user to enlarge or reduce the size of the entire plot by specifying the ratio of the desired plot size to the normal plot size. A sample FACTOR statement is shown in Figure 2-2.

CALL FACTOR (FACT)

FACT
is the ratio of the desired plot size to the normal plot size. For example, if FACT = 2.0, all subsequent pen movements will be twice their normal size. When FACT is reset to 1.0, all plotting returns to normal size.

```
70   CALL PLOT(-0.5,-0.5, -3)
     Positions the pen at the initial
     origin and defines this as the
     new origin.
```




```
80   CALL FACTOR(0.5)
     Reduces the following pen
     movements by 1/2.

90   CALL PLOT(2.0,2.0,2)

100  CALL PLOT(20.0, 0.0, 999)
     Advances chart by 10.0 cms
     and closes output file.
```

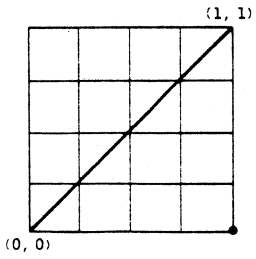


Figure 2-2. Examples of PLOT and FACTOR

2.4 WHERE SUBROUTINE

The WHERE subroutine places the value of the current X and Y coordinates into the variables RXPAGE and RYPAGE respectively and places the current plot size factor into RFACT. This permits user-written subroutines to determine the pens current location for optimizing pen movement.

The calling sequence has three arguments:

CALL WHERE (RXPAGE, RYPAGE, RFACT)

RXPAGE, RYPAGE
Are variables that will be set to the pen's current position coordinates. These coordinates are a result of a previous call to PLOT (which may have been called by SYMBOL, NUMBER, CIRCLE, AXIS or LINE).

RFACT
is set to the current plot size factor; i.e. the value last supplied by a call to FACTOR or 1.0 if FACTOR has not been called.

2.5 NEWPEN SUBROUTINE

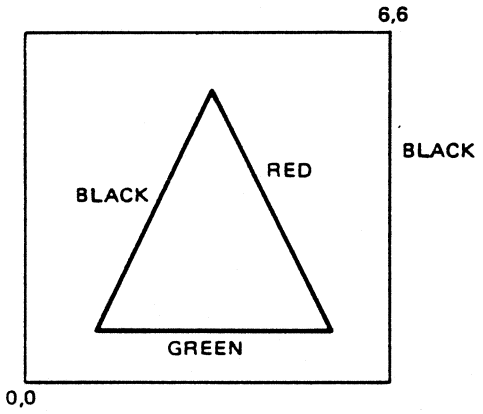
The NEWPEN subroutine enables program selection of any of the available eight pens. PEN 1 is initially selected when the PLOTS routine is called.

The calling sequence is as follows:

CALL NEWPEN (INP)

INP
specifies the number of the pen to be selected. The old pen is raised and stored in its stall. The new pen is picked up and moved to the same physical location at which the old pen was positioned.

If INP=0 the old pen is deposited in the pen holder and no new pen selected.




```
CALL WINDOW (0.0, 6.0, 0.0, 6.0, 1)
CALL PLOT (1.0, 1.0, 3)
CALL PLOT (3.0, 5.0, 2)
CALL NEWPEN (2)
CALL PLOT (5.0, 1.0, 2)
CALL NEWPEN (3)
CALL PLOT (1.0, 1.0, 2)
CALL NEWPEN (0)
CALL CHTADV (20)
```

Figure 2-3. Examples of NEWPEN

2.6 SYMBOL SUBROUTINE

The SYMBOL subroutine produces plot annotation at any angle and in practically any size. There are three SYMBOL call formats:

(1) The "standard" call, which can be used to draw text characters and symbols such as titles, captions, and legends;

(2) the "special" call, which is used to draw special centered symbols such as a box, octagon, triangle, etc., for plotting data points; and

(3) the "set mode" call specific to the Model 81 Plotter. This call is used:

a) to select one of the five Model 81 character fonts or the optional special character font;

b) enable italic mode characters;

c) define a variable aspect ratio for characters.

All forms of the SYMBOL calling sequence have six arguments.

The standard characters that are drawn by SYMBOL include the letters A-Z, digits 0-9, and certain special characters, as detailed in Figure 2-4. The exact availability of all of these symbols may also depend on the computer being used.

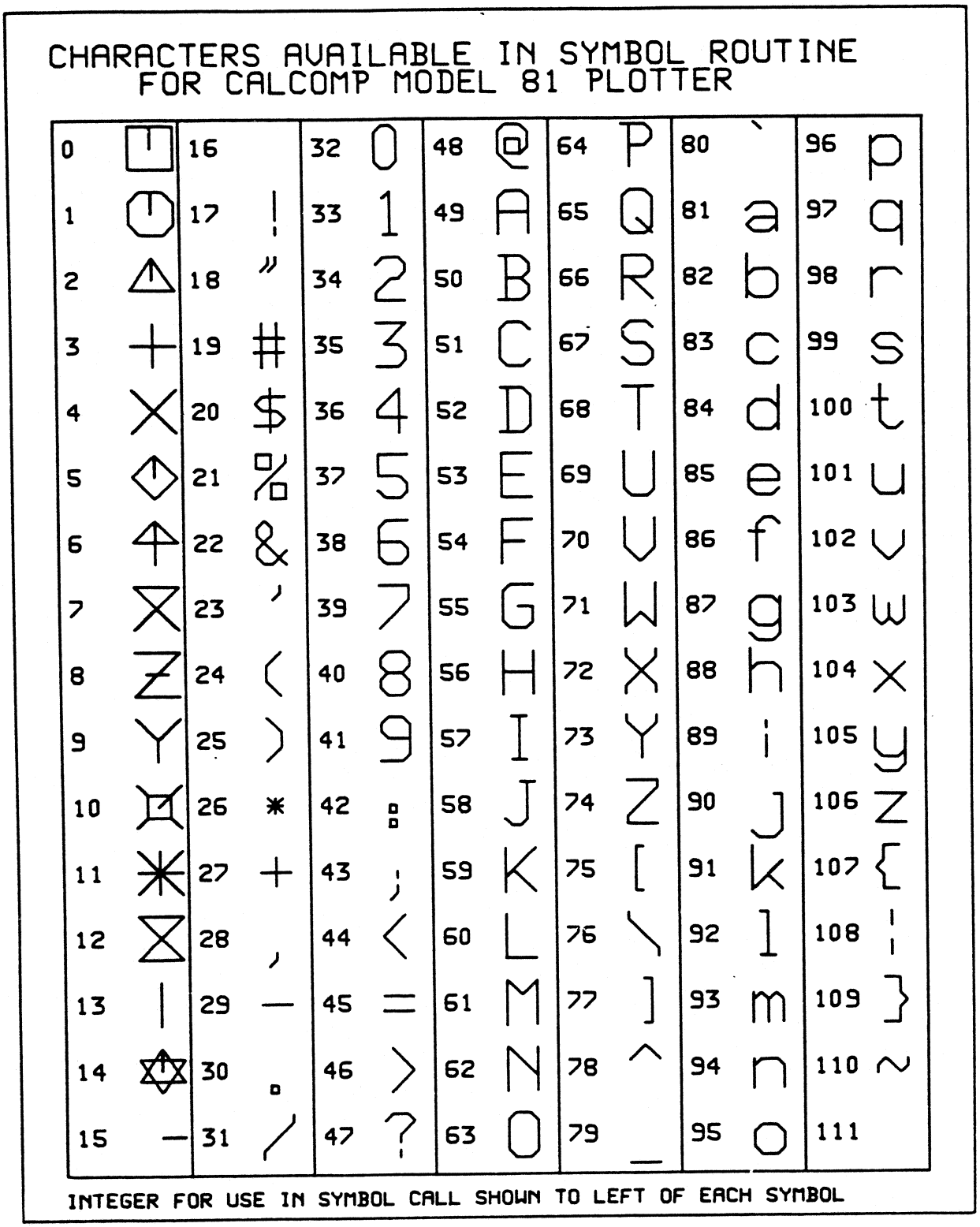


Figure 2-4. Model 81 ASCII Character Set

2.6.1 STANDARD SYMBOL CALL

The standard call draws one or more characters as a text string at a defined location and angle.

The standard call is:

CALL SYMBOL (XPAGE, YPAGE, HEIGHT, IBCD, ANGLE, +NCHAR)

XPAGE, YPAGE
are the coordinates, in cms of the lower left-hand corner (before character rotation) of the first character to be produced. The pen is up while moving to this point.

Annotation may be continued from the last character plotted. Continuation occurs when XPAGE and/or YPAGE equals 999.0, and may be applied to X or Y independently.

HEIGHT
is the height, in centimeters, of the character to be plotted. For best results, it should be a multiple of six times the plotter increment size of 0.1mm: other values are acceptable but are rounded to multiples of 0.6mm after scaling by the current factor. The width of a character, including spacing, is normally the same as the height (e.g. a string of 10 characters 0.24 cms high is 2.4 cms wide). Refer to the "set mode" form of the SYMBOL call.

The characters are designed on a 7 x 7 matrix, two units being used for inter-character spacing. (Examples of character design are given in Figure 2-5).

IBCD
is the text, in internal computer representation (usually A-type format), to be used as annotation. The character(s) must be left-justified and contiguous in a single variable, an array, or a Hollerith literal (if the compiler permits). The characters must only be printable characters. Other characters will cause a loss of origin within the plot job.

If a single character is desired and NCHAR = 0 the text must be right-justified in IBCD.

ANGLE
is the angle, in degrees from the horizontal at which the annotation is to be plotted. If ANGLE = 0.0, the character(s) will be plotted left to right and parallel to the X-axis.

ANGLE should be a whole number. Other values are acceptable but will be rounded to a whole number of degrees.

+NCHAR
is the number of characters to be plotted from IBCD. If NCHAR is greater than 0, the data must be left-justified in each element of IBCD.

If NCHAR = 0, one alphanumeric character is produced, using a single character which is right-justified in the first element of IBCD.

The maximum value of NCHAR must be two less than the length of the I/O buffer.

Some examples of using the "standard" call to SYMBOL are shown in Figure 2-6.

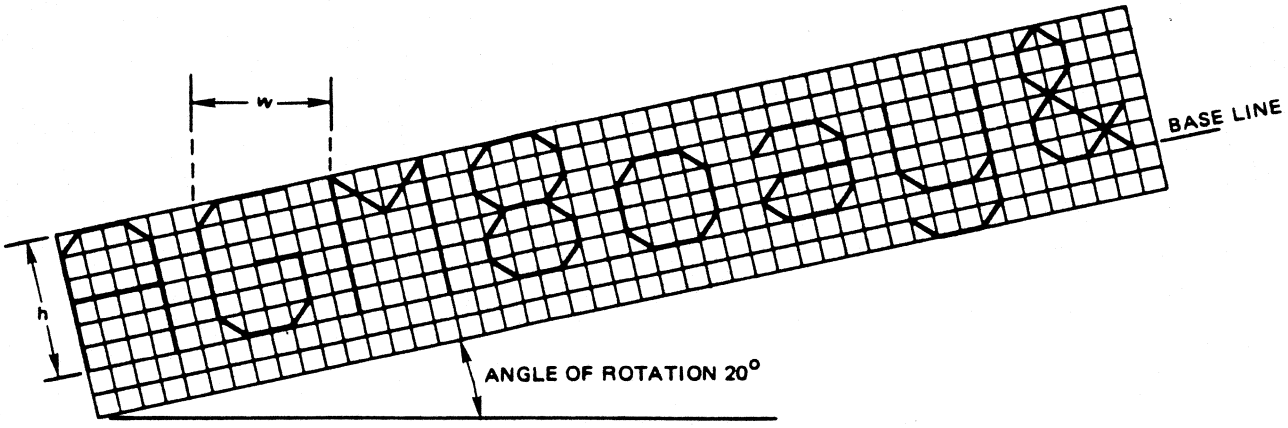


Figure 2-5. Character Design

```
DIMENSION IBCD (8)
CALL SYMBOL (X, Y, 0.9, IBCD, 0., 16)

NOTE:  IBCD IS AN 8-WORD ARRAY CONTAINING
       16 CHARACTERS
```

```
CALL SYMBOL (X, Y, 0.3, IBCD, 0.0, 16)
CALL SYMBOL (999.0, 999.0, 0.6, IBCD, 90.0, 16)
CALL SYMBOL (999.0, 999.0, 0.6, IBCD, 180.0, 16)
CALL SYMBOL (999.0, 999.0, 0.6, IBCD, 270, 0.16)

NOTE:  TWO SPACES FOLLOW "A SAMPLE TITLE"
```

A SAMPLE TITLE

A SAMPLE TITLE A SAMPLE TITLE A SAMPLE TITLE A SAMPLE TITLE

Figure 2-6. Examples of Standard SYMBOL

2.6.2 SPECIAL SYMBOL CALL

The second form is the "special" call, which produces only a single symbol based on the integer value of INTEQ and is identified by a negative value of ICODE.

The "special" call sequence is:

CALL SYMBOL (XPAGE, YPAGE, HEIGHT, INTEQ, ANGLE, -ICODE)

XPAGE, YPAGE, and ANGLE
are the same as described for the "standard" call. If the symbol to be produced is one of the centered symbols (i.e. if INTEQ is less than 15), XPAGE, YPAGE represent the geometric center of the character produced.

HEIGHT
is the height (and width) in centimeters, of the centered symbol to be drawn. Preferably, it should be a multiple of four times the plotter's increment size for centered symbols.

INTEQ
is the integer equivalent of the desired symbol. Valid integers and their symbols are listed in figure 2-4. If INTEQ is 0 through 15, a centered symbol is produced.

-ICODE
must be negative and determines whether the pen is up or down during the move to XPAGE, YPAGE.

WHEN -ICODE is:
-1, the pen is up during the move, after which a single symbol is produced.

-2, or less, the pen is down during the move, after which a single symbol is produced.

```
CALL SYMBOL (3.0, 1.0, 0.3, 3, 0.0, -2)
CALL SYMBOL (6.0, 0.0, 0.3, 9, 0.0, -2)
CALL SYMBOL (9.0, 0.0, 0.3, 2, 0.0, -2)
```

Figure 2-7. Example of Special SYMBOL

2.6.3 SET MODE SYMBOL CALL

The third form is the "set mode" call, which does not produce annotation at the time of the call but establishes the mode of operation of the symbol routine for subsequent calls. This form of the SYMBOL call is identified by a negative value of HEIGHT.

The "set mode" call is:

CALL SYMBOL (ASPECT, TALIC, -HEIGHT, IBCD, SPACE, IFONT)

ASPECT
is the ratio of character envelope width to height. Width refers to the distance from the lower left corner of one character envelope to the lower left corner of the next character envelope, thus including the inter-character spacing. Height refers to the height of the character. (see figure 2-5)

ASPECT=1.0 for normal annotation
ASPECT >1.0 for broad (short, fat) annotation
ASPECT< 1.0 for narrow (tall, thin) annotation
ASPECT=999.0 for no ASPECT ratio change. The value of ASPECT from the previous "set mode" call will remain in effect.

TALIC
determines the slant (italicization) of characters. Normally, TALIC=0.0 for vertical letters. If TALIC is non-zero characters are drawn at a 75 degree slant to the right relative to the base line of the character string. If TALIC=999.0 the value of TALIC from the previous "set mode" call remains in effect.

HEIGHT
when HEIGHT is negative, the modifying parameters ASPECT, TALIC and IFONT can be set. The magnitude of HEIGHT is not used. To maintain compatibility with other versions of the call, HEIGHT should be set to -999.0.

IBCD
is reserved for future use.

SPACE
is reserved for future use.

IFONT
determines the character font to be used. Characters available with each font are given in Figure 2-8.

IFONT=0 normal ASCII
IFONT=1 German Font
IFONT=2 Spanish Font
IFONT=3 Swedish/Finnish Font
IFONT=4 Danish/Norwegian Font
IFONT=-1 Special Optional Font*
IFONT=999 for no font change. The value of IFONT from the previous 'set mode' call will remain in effect.

Examples of 'set mode' call are given in Figure 2-9.

NOTE:

Initial values of 'set mode' parameters ASPECT, TALIC and IFONT are:

ASPECT = 1.0
TALIC = 0.0
IFONT = 0

To return to a normal mode of symbol output:

CALL SYMBOL (1.0, 0.0, -1.0, 999.0 999.0, 0)

Centered symbols (INTEQ of 0 through 15) are always drawn with an aspect ratio of 1.0 and no slant.

---

* Extra cost factory option.

FONT NO. 0 (STANDARD ASCII)

```
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
```

FONT NO. 1 (GERMAN)

```
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ^_
`abcdefghijklmnopqrstuvwxyzäöü~
```

FONT NO. 2 (SPANISH)

```
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZiÑ¿^_
`abcdefghijklmnopqrstuvwxyz{|}~
```

FONT NO. 3 (SWEDISH – FINNISH)

```
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÅ^_
`abcdefghijklmnopqrstuvwxyzäöå~
```

FONT NO. 4 (DANISH – NORWEGIAN)

```
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZÆØÅ^_
`abcdefghijklmnopqrstuvwxyzæøå~
```

Figure 2-8. Model 81 Character Sets

1. CALL SYMBOL (2.0, 0.0, -1.0, IDUM, DUMY, 0)
   CALL SYMBOL (0.0, 0.0, 0.6, IBCD, 0.0, 11)

   (WHERE IBCD CONTAINS THE HOLLERITH STRING "TEXT STRING")

TEXT STRING

2. CALL SYMBOL (999.0, 1.0, -1.0, IDUM, DUMY, 0)
   CALL SYMBOL (0.0, 0.0, 0.6, IBCD, 0.0, 11)

TEXT STRING

3. CALL SYMBOL (1.0, 0.0, -1.0, IDUM, DUMY, 0)
   CALL SYMBOL (0.0, 0.0, 0.6, IBCD, 0.0, 11)

TEXT STRING

Figure 2-9. Examples of Set Mode SYMBOL

2.7 NUMBER SUBROUTINE

The NUMBER subroutine converts a floating-point number to the appropriate ASCII character equivalent so that the number may be plotted.

The NUMBER calling sequence has six arguments:

CALL NUMBER (XPAGE, YPAGE, HEIGHT, FPN, ANGLE, ±NDEC)

XPAGE, YPAGE, HEIGHT & ANGLE
are the same as those arguments described for subroutine SYMBOL. The continuation feature, where XPAGE or YPAGE equals 999.0, may also be used.

FPN
is the floating-point number that is to be converted and plotted.

±NDEC
controls the precision of the conversion of the number FPN. If the value of NDEC>0, it specifies the number of digits to the right of the decimal point that are to be converted and plotted, after rounding. For example, assume an internal value of

$$-0.12345678 \times 10^{3}.$$

If NDEC = 2, the plotter number would be -123.45.

If NDEC = 0, only the number's integer portion and a decimal point are plotted, after rounding.

If NDEC = -1, only the number's integer portion is plotted, after rounding. (The above example would be plotted as -123 with no decimal point).

If NDEC <-1, |NDEC| -1 digits are truncated from the integer portion, after rounding.

The magnitude of NDEC should not exceed 9.

Figure 2-10 illustrates various uses of SYMBOL and NUMBER.

```
COMBINING SYMBOL AND NUMBER AND DRAWING
A SUPERSCRIPT

CALL SYMBOL (X, Y, 0.36, 10HVALUE OF X, 0., .0)
CALL SYMBOL (999., Y+0.2, 0.18, 2H2 , 0., 2)        SUPERSCRIPT
CALL SYMBOL (999., Y, 0.36, 2H= , 0. , 2)
CALL NUMBER (999., 999., 0.36, VALUE, 0. , 3)
```

VALUE OF X^2= 12.123

```
DRAWING TEST AND NUMBERS AT VARIOUS ANGLES

D0 10 J=0, 315, 45
ANGLE = J
CALL SYMBOL (X, Y, 0.24, 7H ANGLE = , ANGLE, 7)
CALL NUMBER (999., 999., 0.24, ANGLE, ANGLE, -1)
```

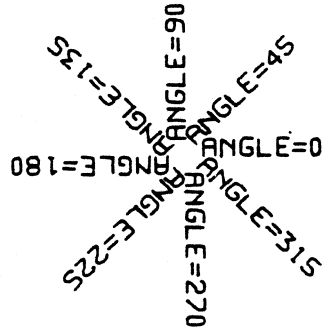


Figure 2-10. Example of SYMBOL and NUMBER

2.8 SCALE SUBROUTINE

Typically, the user's program will accumulate plotting data in two arrays:

- An array of independent variables, X_i
- An array of dependent variables, $Y_i = f(X_i)$

It would be unusual if the range of values in each array corresponded exactly with the number of centimeters available in the actual plotting area. For some problems the range of data is predictable. The programmer can predetermine suitable conversion factors for use in drawing the axis scale values and plotting the data points on the graph. Usually, however, these factors are not known in advance.

The SCALE subroutine is used to examine the data values in an array and to determine a starting value (minimum or maximum) and a scaling factor (positive or negative) such that:

1) The scale annotation drawn by the AXIS subroutine at each division will properly represent the range of real data values in the array; and

2) The data points, when plotted by the LINE subroutine, will fit in a given plotting area. These two values are computed and stored by SCALE at the end of the array.

The scaling factor (DELTAV) that is computed represents the number of data units per cm of axis but is adjusted so that it is always an interval of 1, 2, 4, 5, or 8×10^n (where n is an exponent consistent with the original unadjusted scaling factor). For example, an array may have a range of values from 301 to 1512, to be plotted over an axis of 20 cm. The unadjusted scaling factor is (1512-301)/20 = 57.7 units/cm. The adjusted DELTAV would be 8×10^1 = 80 units/cm.

The starting value (FIRSTV), which will appear as the first annotation on the axis, is computed as some multiple of DELTAV that is equal to or outside the limits of the data in the array. For the example given above, if a minimum is wanted for FIRSTV, 240 would be chosen as the best value. If a maximum is desired instead, 1520 would be selected. In some instances, FIRSTV is selected as a downward-rounded value of the lowest actual data, and in other instances, the DELTAV is adjusted upwards. An attempt is then made to center the data.

There are four arguments in the calling sequence:

CALL SCALE (ARRAY, AXLEN, NPTS, ± INC)

ARRAY
is the name of the array of data points to be examined.

AXLEN
is the length of the axis to which the data is to be scaled. Its value must be greater than 2.0 cms.

NPTS
is the number of data values to be scanned in the array. The FORTRAN DIMENSION statement must specify at least two elements more than the number of values being scanned to allow room for SCALE to store the computed starting value and scaling factor at the end of the array.

±INC
is an integer whose magnitude is used by SCALE as the increment with which to select the data values to be scanned in the array. If INC = 1, every value in the array is examined. If INC = 2, every other value is examined.

If INC is positive, the selected starting value (FIRSTV) is less than the minimum, and the scale factor (DELTAV) is positive.

If INC is negative, the selected starting value (FIRSTV) is greater than the maximum, and the scaling factor (DELTAV) is negative.

If INC is plus or minus 1, the array must be dimensioned at least two elements larger than the actual number of data values it contains. If the magnitude of INC>1, the computed values are stored at (INC) elements and (2*INC) elements beyond the last data point. The subscripted element for FIRSTV is ARRAY (NPTS*INC+1); for DELTAV it is ARRAY (NPTS*INC+INC+1).

Generally, SCALE is called to examine each array to be plotted, as shown earlier in A SAMPLE PLOTTING PROGRAM. If the user knows the range of his data values, he does not have to call SCALE for that array so long as he supplies an appropriate FIRSTV and DELTAV when AXIS and LINE are called.

Figure 2-11 illustrates some typical uses of SCALE.

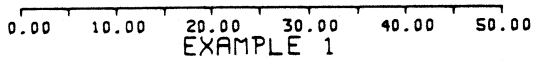
EXAMPLE 1:

GIVEN AN ARRAY OF 24 DATA VALUES TO BE PLOTTED OVER A 10 CM AXIS, ASSUME THE MINIMUM VALUE IN THE ARRAY IS 1.00 AND THE MAXIMUM IS 42.00. THE STATEMENT

CALL SCALE (ARRAY, 10.0, 24, +1) WOULD GIVE THE FOLLOWING RESULTS:

UNITS/CM = (42.00 -1.00)/10.0 = 4.1
DELTAV (NEXT HIGHER INTERVAL) = 5.0
FIRSTV (MINIMUM MULTIPLE) = 0.00

FIRSTV VALUE IS STORED IN ARRAY (25)
DELTAV VALUE IS STORED IN ARRAY (26)
USING THESE VALUES, AXIS WOULD DRAW THE FOLLOWING AXIS LINE:



EXAMPLE 1

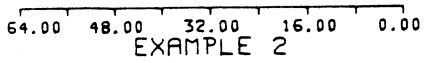
EXAMPLE 2:

ASSUME THAT THE ARRAY IN EXAMPLE 1 IS TO BE PLOTTED ON A 8CM AXIS, FROM MAXIMUM TO MINIMUM.

CALL SCALE (ARRAY, 8.0, 24, -1) WOULD GIVE THESE RESULTS.

DELTAV = (1.00 -42.00)/8.0 = -5.125, WHICH IS ADJUSTED TO -8.0
MINIMUM MULTIPLE = 0.00; FIRSTV = MINIMUM + (AXLEN*|DELTAV|) = 64.0.

IN THIS EXAMPLE, THE FOLLOWING AXIS WOULD BE DRAWN:



EXAMPLE 2

Figure 2-11. Examples of SCALE

2.9 AXIS SUBROUTINE

Most graphs require axis lines and scales to indicate the orientation and values of the plotted data points. The most common type of scaled axis is produced by the AXIS subroutine, which draws any length line at any angle, divides the line into one-centimeter segments, annotates every alternate division with appropriate scale values and labels the axis with a centered title. When both the X and Y axes are needed, AXIS is called separately for each one.

The eight arguments in the calling sequence are as follows:

CALL AXIS (XPAGE, YPAGE, IBCD, ± NCHAR, AXLEN, ANGLE, FIRSTV, DELTAV)

XPAGE, YPAGE

are the coordinates, in centimeters, of the axis line's starting point. The line should be at least two centimeters from any side to allow space for the scale annotation and axis title. Usually, both the X and Y axes are joined at the origin of the graph, where XPAGE and YPAGE equal zero; but other starting points can be used. When using the LINE subroutine to plot data on an axis, at least one of the coordinates must be 0, i.e. for an X-axis, XPAGE = 0, and for a Y-axis, YPAGE = 0.

IBCD

is the title, which is centerd and placed parallel to the axis line. This parameter may be an alphanumeric array, or it may be a Hollerith literal if the FORTRAN compiler being used permits it. The characters have a fixed height of 0.30 cms.

± NCHAR

The magnitude specifies the number of characters in the axis title, and the sign determines on which side of the line the scale (tick) marks and labelling information shall be placed. Since the axis line may be drawn at any angle, the line itself is used as a reference.

If the sign is positive, all annotation appears on the positive (counter clockwise) side of the axis. This condition is normally desired for the Y-axis.

If the sign is negative, all annotation appears on the negative (clockwise) side of the axis. This condition is normally desired for the X-axis.

AXLEN

is the length of the axis line, in centimeters.

ANGLE

is the angle in degrees (positive or negative), at which the axis is drawn. The normal value is 0° for the X-axis and 90° for the Y-axis.

FIRSTV

is the starting value (either minimum or maximum) which will appear at the first tick mark on the axis. This value may either be computed by the SCALE subroutine and stored at subscripted location ARRAY (NPTS*INC+1), in which case FIRSTV = ARRAY (NPTS *INC + 1) or the value may be determined by the user and stored at any location. It will appear at the first tick mark on the axis.

This number is drawn with two decimal places. The digit size is 0.24 cms wide. Since a scale value appears every other centimeter, no more than six digits and a sign should appear to the left of the decimal point.

DELTAV

represents the number of data units per centimeter of axis. This value (increment or decrement) is added to FIRSTV for each succeeding one-centimeter division along the axis. This value may either be computed by SCALE and stored at ARRAY (NPTS*INC+INC+1), in which case DELTAV = ARRAY (NPTS*INC+INC+1) or the value may be determined by the user and stored at any location.

In order to use a standard format of two decimal places, the size of DELTAV is adjusted to less than 100, but not less than 0.01. As a result, the decimal point may be shifted left or right in the scale values as drawn, and the axis title is then followed by "$*10^n$", where n is the power-of-ten adjustment factor. (See X-axis example in Figure 2-12).

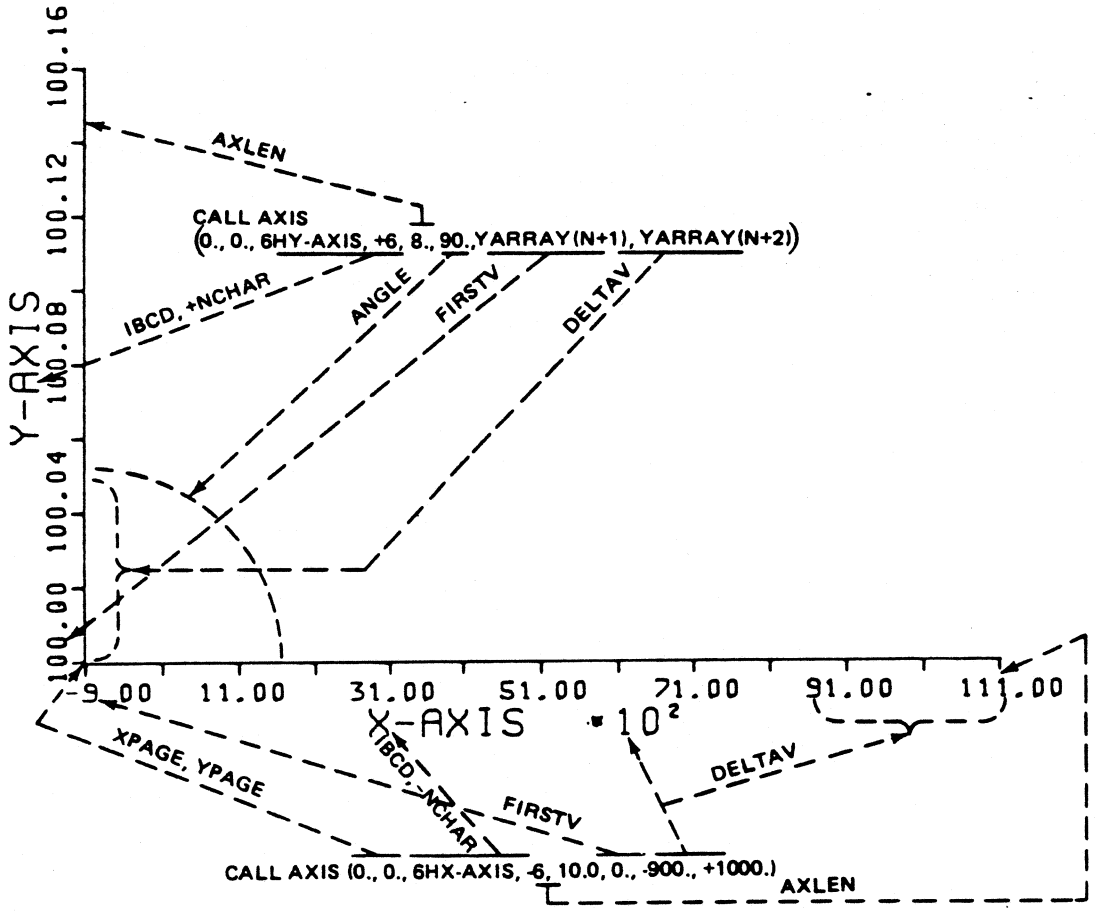


Figure 2-12. Examples of AXIS

2.10 LINE SUBROUTINE

The LINE subroutine produces a line plot of the pairs of data values in two arrays (X and Y). LINE computes the coordinates of each plotted point according to the data values in each array and the respective scaling parameters. The data points may be represented by centerd symbols and/or connecting lines between points.

The scaling parameters corresponding to FIRSTV and DELTAV (see SCALE) must immediately follow each array. If these parameters have not been computed by the SCALE subroutine they must be supplied by the user.

The calling sequence has six arguments:

CALL LINE (XARRAY, YARRAY, NPTS, INC, ± LINTYP, INTEQ)

XARRAY
is the name of the array containing the abscissa (X) values and the scaling parameters for the X-array.

YARRAY
is the name of the array containing the ordinate (Y) values and the scaling parameters for the Y-array.

NPTS
is the number of data points in each of the two array just mentioned. The number does not include the extra two locations for the scaling parameters. The number of points in each array must be the same.

INC
is the increment that the LINE subroutine is to use in gathering data from the two arrays, as described previously for the SCALE subroutine.

±LINTYP
is a control parameter which describes the type of line to be drawn through the data points. The magnitude of LINTYP determines the frequency of plotted symbols, e.g. if LINTYP = 4, a symbol (denoted by INTEQ) is plotted at every fourth data point.

If LINTYP is zero, the points are connected by straight lines but no symbols are plotted.

If LINTYP is positive, a straight line connects every data point defined in the array. A symbol is plotted at points specified by the magnitude of LINTYP. If 1, a symbol is drawn at each point; if 2, at every other point, etc. (The pen is up when moving from its current position to the first point).

If LINTYP is negative, no connecting lines are drawn; only the symbols are plotted.

INTEQ
is the integer equivalent of the plotting symbol centered at each data point. This value normally can be 0 through 15 and has meaning only when LINTYP is not zero. Figure 2-4 lists the symbols that are available. The centered symbols have a fixed height of 0.2 cms. Figure 2-13 illustrates the types of line drawn by various combinations of LINTYP and INTEQ.

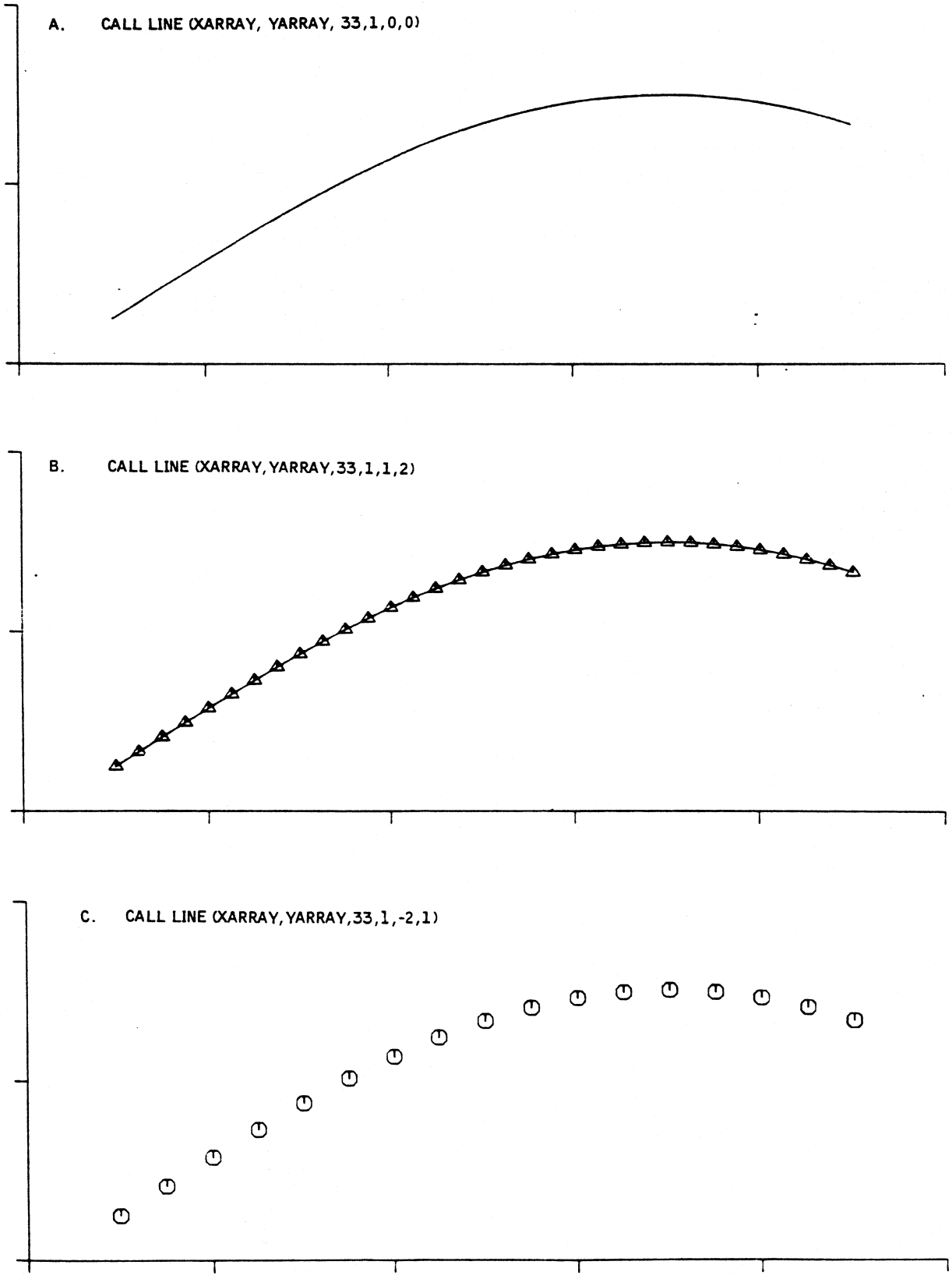


Figure 2-13. Example of LINE

2.11 DASHS SUBROUTINE

The DASHS Subroutine is used when patterned lines are desired in place of the standard continuous line. Four patterns in addition to the continuous line are selectable through this subroutine. The pattern type and length can be specified to add variety to plots and as an aid in distinguishing data.

When dashline is enabled, all subsequent calls to PLOT, CIRCLE or any other subroutines which call PLOT will produce dashlines as determined by the previous call to DASHS. Symbols are always drawn with a solid line.

The calling sequence has two arguments:

CALL DASHS (ARRAY, ICNT)

ARRAY
is an array containing two floating-point values used to select the dashline type and define the pattern length. When ICNT is zero, ARRAY is not used.

ARRAY(1)
is a whole real number of the value 0.0, 1.0, 2.0, 3.0, or 4.0 used to define the dashline type.

ARRAY (1)=1.0 dotted line
ARRAY (1)=2.0 dashed line
ARRAY (1)=3.0 unequaled dash line
ARRAY (1)=4.0 dash-dot line

If ARRAY(1) equals zero the previously defined dashline type and 'pattern' length are used and ARRAY(2) is not used. See Figure 2-14.

ARRAY(2)
is the length of the dashline pattern in centimeters.

ICNT
is a flag used to enable the dashline mode.

If ICNT=0, all lines drawn are standard continuous lines.

If ICNT=+1, the dashline mode is turned on and the dashline type and pattern length are defined by ARRAY.

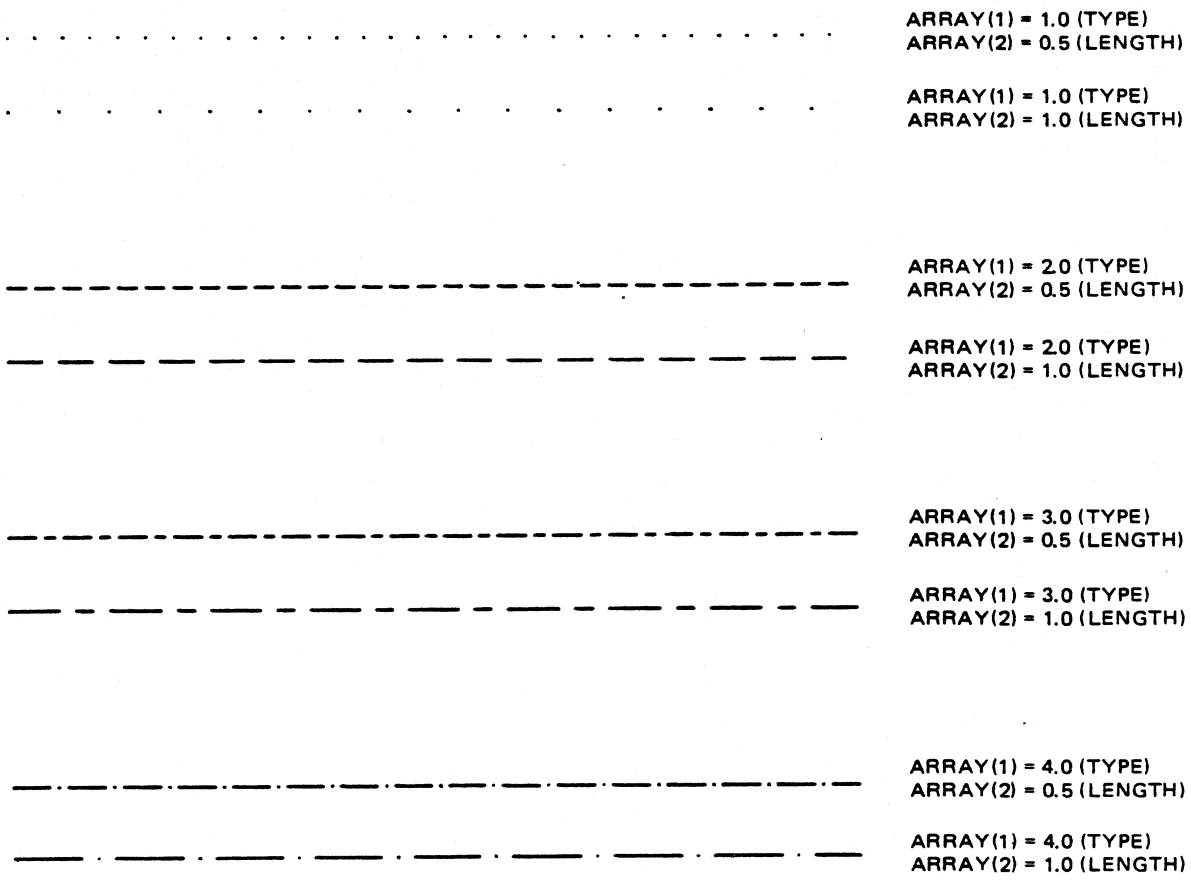


Figure 2-14. Examples of DASHS

2.12 CIRCLE SUBROUTINE

An arc (or circle, which is a 360° arc) may be specified in several different ways using different characteristics of a circle. Some of these characteristics are readily known from the drawing, others may require calculation to convert from cartesian to polar coordinates. The circle subroutine accommodates a variety of methods depending upon the characteristics known.

The simplest specification is a circle. The center of the circle and one point on the circle must be specified in cartesian coordinates as illustrated in figure 2-15.

If the starting point, ending point, radius and direction is know, method 2 or 3 may be used as shown in figure 2-16. In this method two different center points can be derived from the points given and therefore two arcs are possible. The desired arc is chosen by selecting the minor arc (less than 180°), method 2 or the major arc (greater than 180°), method 3. This method is useful when an arc must connect two known points.

Another method uses the starting point, starting angle, ending angle and the radius as illustrated in figure 2-17. This method is useful when drawing corners to connect straight lines in either direction. Mode 4 is used for circles, mode 5 for arcs.

A third method uses the starting angle, ending angle and radius as in method 5, but uses the center point instead of the starting point for reference. This method may be used when connecting lines and arcs around a centerline as shown in figure 2-18. Mode 6 is used for circles, mode 7 for arcs.

The calling sequence has six arguments:

CALL CIRCLE (P1, P2, P3, P4, P5, IMODE)

CIRCLE Subroutine Modes

Function	P1	P2	P3	P4	P5	IMODE	Pen Position at Exit
Full Circle	Xs	Ys	Xc	Yc		±1	Start of Circle
Arc (Minor)	Xs	Ys	Xe	Ye	R	±2	End of Arc
Arc (Major)	Xs	Ys	Xe	Ye	R	±3	End of Arc
Full Circle	Xs	Ys	As		R	±4	Start of Circle
Arc	Xs	Ys	As	Ae	R	5	End of Arc
Full Circle	Xc	Yc	As		R	±6	Start of Circle
Arc	Xc	Yc	As	Ae	R	7	End of Arc

Where: (Xs,Ys) is the starting point of the circle/arc
(Xe,Ye) is the ending point of the arc
(Xc,Yc) is the center of the circle/arc
R is the radius of the circle/arc
As is the starting angle (all angles are measured counterclockwise from the horizontal)
Ae is the ending angle
(If Ae > As, an arc is drawn from As to Ae in a counterclockwise direction; if Ae < As, an arc is drawn in a clockwise direction.)

IMODE is the type of circle/arc
(positive = counterclockwise, negative = clockwise, where applicable)

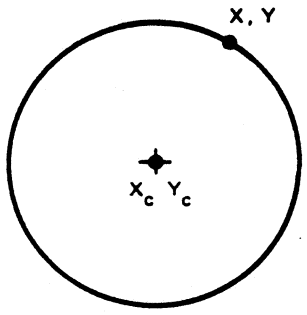
NOTES:

P1, P2, P3, P4 & P5 are floating-point values. IMODE is an integer value.

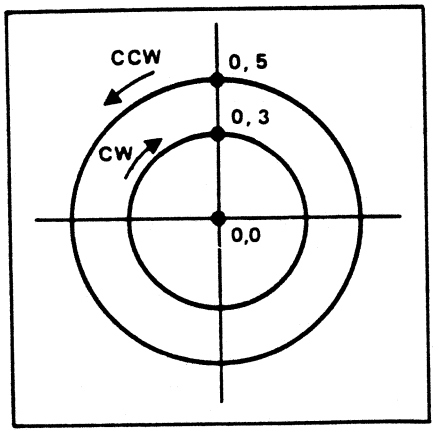
The major arc is the longer distance from a point to the end point of an arc. Conversely, the minor arc is the shorter distance from the same point to the end point of an arc.

The maximum radius allowed for a circle is 32,767 plotter increments (227 cm).

Dashed circles/arcs may be drawn by calling the DASHS routine prior to calling the CIRCLE routine.

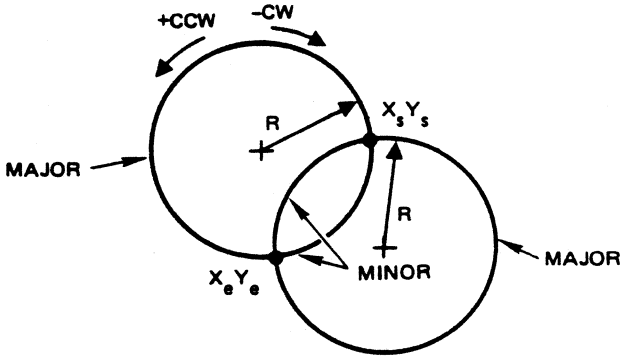


CALL CIRCLE (XS, YS, XC, YC, 0.0, ±1)

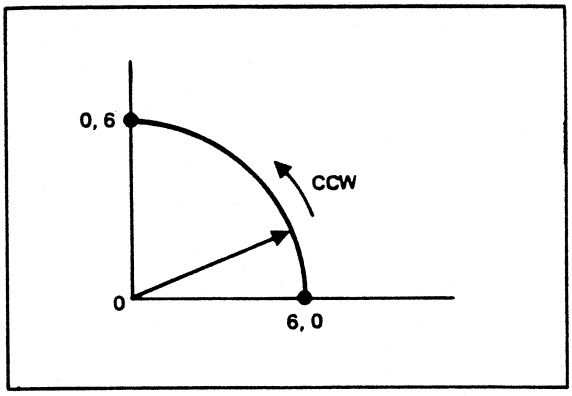


CALL CIRCLE (0.0, 3.0, 0.0, 0.0, 0.0, 1)
CALL CIRCLE (0.0, 5.0, 0.0, 0.0, 0.0, -1)

Figure 2-15. Example of CIRCLE Mode 1

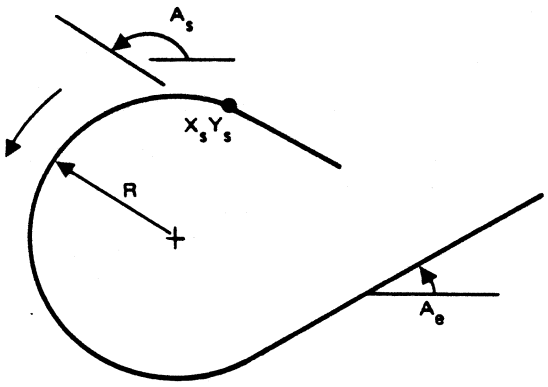


CALL CIRCLE (XS, YS, XE, YE, R, ±2) MINOR ALL
CALL CIRCLE (XS, YS, XE, YE, R, ±3) MAJOR ALL



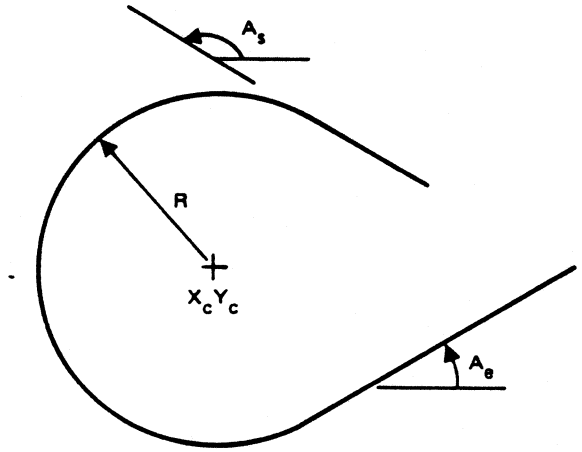
CALL CIRCLE (6.0, 0.0, 0.0, 6.0, 6.0, 2)

Figure 2-16. Example of CIRCLE Modes 2 & 3

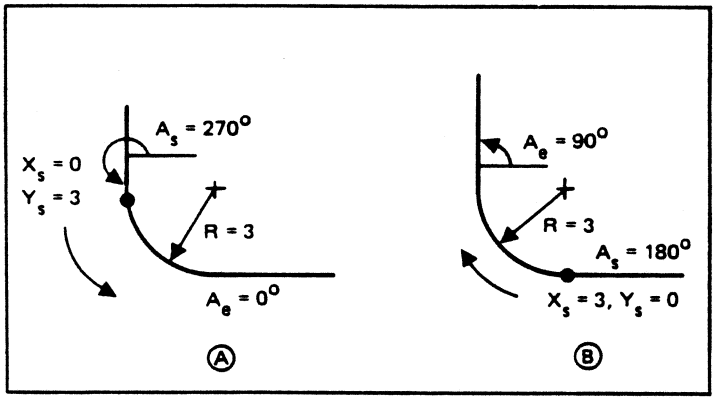


CALL CIRCLE (XS, YS, AS, AE, R, 5) ARC
CALL CIRCLE (XS, YS, AS, 0.0, R, ± 4) CIRCLE

Figure 2-17. Example of CIRCLE Modes 4 & 5

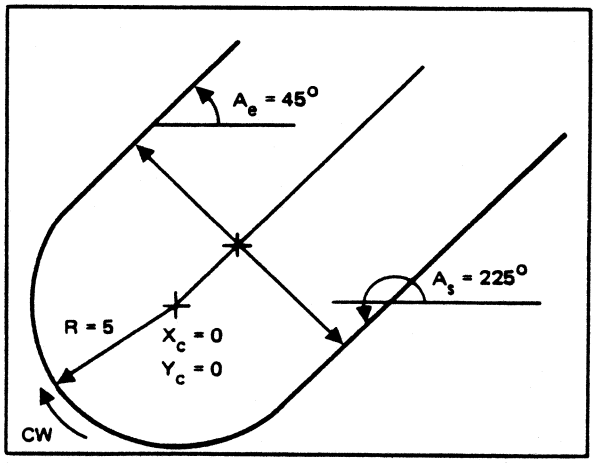


CALL CIRCLE (XC, YC, AS, AE, R, 7) ARC
CALL CIRCLE (XC, YC, AS, 0.0, R, ± 6) CIRCLE



Ⓐ CALL CIRCLE (0.0, 3.0, 270.0, 0.0, 3.0, 5)
Ⓑ CALL CIRCLE (3.0, 0.0, 180.0, 90.0, 3.0, 5)

Figure 2-18. Example of CIRCLE Modes 6 & 7



CALL CIRCLE (0.0, 0.0, 225.0, 45.0, 5.0, 7)

.13 WINDOW SUBROUTINE

he WINDOW subroutine is specific to the lodel 81 Plotter. The routine allows the ser to define an inclusive rectangular 'indow for plotting such that only those ortions of the plot job that fall within the pecified window limits will be plotted. 'he user may also verify the defined /indow by plotting the window boundary.

'he WINDOW calling sequence has five .rguments:

:ALL WINDOW (XMIN, XMAX, YMIN, YMAX, IVFLG)

(MIN, YMIN
define the X and Y coordinates of the lower left corner of the rectangle to be used as a window.

XMAX, YMAX
define the X and Y coordinates of the upper right corner of the rectangle to be used as a window.

IVFLG
is a flag used to indicate whether a rectangular frame should be drawn to indicate the window boundary.

IVFLG = 0, no frame
IVFLG 0, draw window boundary

NOTES:

A window may not be defined that has limits outside of the Model 81 plotting surface. Window coordinates are in terms of user coordinates. Two examples are given in Figure 2-19.

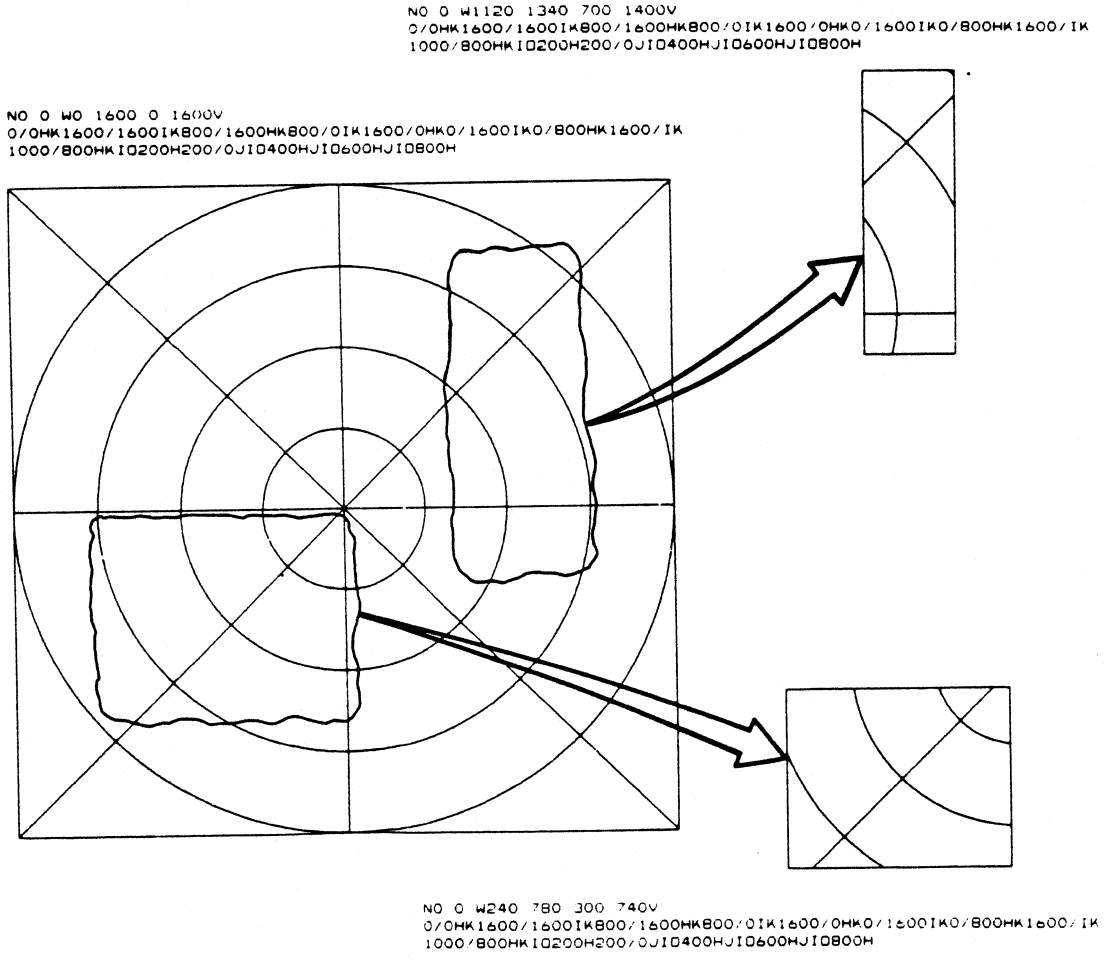


Figure 2-19. Example of Window

2.14 CHTADV SUBROUTINE

A Roll Paper Advance option is available for the Model 81 plotter. When attached, a roll of paper, 25m in length can be mounted on the supply roller and advanced across the platen manually or under program control.

The CHTADV subroutine is specific to the Model 81 Plotter.

The calling sequence has one argument:

CALL CHTADV(LEN)

LEN
is the distance in centimeters to advance the paper.

If LEN is positive the value specified in LEN is in user units. The actual distance the paper will be advanced is LEN times the scale FACTOR, rounded to the nearest whole centimeter.

If LEN is negative the paper is advanced LEN centimeters.

The maximum distance the paper may be advanced is 63 cms.

3. OPERATING CONSIDERATIONS

An understanding of some of the operating characteristics of the CalComp Model 81 Plotter is desirable when planning an application program. The quality of the finished plot depends on the combination of pen, paper and ink.

The various types and care of these items are described in the Model 81 Operator Manual P/N 10255-901-001. The accuracy of the plot is affected by the origin and scale set on the operator panel.

When PLOTS is called, a logical origin is defined by setting the current X,Y coordinates maintained in the PLOT routine to zero. When actual plotting begins, this logical origin corresponds to the current physical pen position of Pen 1, which is established as the physical origin. Under initial conditions this position corresponds to the bottom left limit of the plotting area. Before a plot is started, the operator may move the pen to a position which is suitable for the origin of the plot. (Ref. Model 81 Operators Manual). Typically this is the lower left-hand corner of the area to be occupied by the plot, but it will depend upon the way the plot was programmed.

If gridded paper or special pre-printed forms are used, the operator must align the forms properly before starting to plot. This can be done by replacing Pen 1 with the crosshair reticle provided in the accessory kit, using the manual positioning pad to move the reticle over the desired point and selecting a new zero origin as described in the Model 81 Operators Manual.

Variable scaling at plot-time is also possible by setting up a new scaling point as described in the Model 81 Operators Manual. The plotter origin and scale point cannot be reset by the Applications Program.

4. COMPATIBILITY

4.1 MODEL 81 BASIC COMMANDS

Integral to the Model 81 is a microprocessor which receives the HCBS generated commands and controls the plotter actions. The capabilities and programming at this level is fully described in the Model 81 Basic Commands Manual P/N 10255-901-007. This basic commands programming is not compatible with any other CalComp software but does offer some additional facilities not incorporated in the HCBS. They are:

- DIGITIZING
- INCREMENTAL PLOTTING
- HORIZONTAL TABBING
- VERTICAL TABBING
- OFFSET/WINDOWING

These functions can be accessed with simple WRITE statements to the plotter however mixing the two programming mode is not recommended. If used concurrently, extreme care must be taken to insure that actions performed in one mode are accounted for in the other.

4.2 CALCOMP SOFTWARE

CalComp provides three HCBS software packages, the Standard HCBS for offline controllers, the 907 HCBS for Model 907 online controllers and the Model 81 HCBS. The Model 81 HCBS is upward compatible with the Standard and 907 HCBS's as far as possible however, because of differences in capabilities some incompatibilities do exist. These are identified and explained in Table 4-1.

TABLE 4-1. HCBS COMPATIBILITIES

SUBROUTINE	M81 HCBS	907 HCBS	STD HCBS
PLOT	A	X	X
PLOTS	X	X	X
FACTOR	X	X	X
WHERE	X	X	E
NEWPEN	X	X	X
SYMBOL	B	C	X
NUMBER	X	X	X
SCALE	X	X	X
AXIS	X	X	X
LINE	X	X	X
DASHS	D	X	O
CIRCLE	X	X	O
WINDOW	X	O	O
CHTADV	X	O	O
USRSYM	O	X	O

X The CALL and parameter specifications are identical and the execution is functionally the same.

O The subroutine is not in the indicated HCBS.

A The IPEN=999 functions is slightly different.

B The Set Mode option of the SYMBOL subroutine is added.

C The SYMBOL subroutine is upgraded to include extended and user defined symbols.

D DASHS pattern types and specifications are different.

E WHERE will return beginning coords. of SYMBOL string. M81 and 907 HCBS will return coords. of end of SYMBOL string.

APPENDIX A

SUMMARY OF CALLS

CALL PLOT (XPAGE, YPAGE, ±IPEN)

CALL PLOTS (0, 0, LDEV)

CALL FACTOR (FACT)

CALL WHERE (RXPAGE, RYPAGE, RFACT)

CALL NEWPEN (INP)

CALL SYMBOL (XPAGE, YPAGE, HEIGHT, IBCD, ANGLE, +NCHAR)

CALL SYMBOL (XPAGE, YPAGE, HEIGHT, INTEQ, ANGLE, -ICODE)

CALL SYMBOL (ASPECT, TALIC, -HEIGHT, IBCD, SPACE, IFONT)

CALL NUMBER (XPAGE, YPAGE, HEIGHT, FPN, ANGLE, ±NDEC)

CALL SCALE (ARRAY, AXLEN, NPTS, ±INC)

CALL AXIS (XPAGE, YPAGE, IBCD, ±NCHAR, AXLEN, ANGLE, FIRSTV, DELTAV)

CALL LINE (XARRAY, YARRAY, NPTS, INC, ±LINTYP, INTEQ)

CALL DASHS (ARRAY, ICNT)

CALL CIRCLE (P1, P2, P3, P4, P5, IMODE)

CALL WINDOW (XMIN, XMAX, YMIN, YMAX, IVFLG)

CALL CHTADV (LEN)

```
SUBROUTINE ERASE

CLEARS THE VIEWPORT
```

```
SUBROUTINE FACTOR(FACT)

SCALE WINDOW WITH FACT
```

```
SUBROUTINE FRAME

DRAWS A FRAME AROUND THE VIEWPORT
```

```
SUBROUTINE GRID(X,Y,XS,YS,M,N)

WHERE -(X,Y) IS THE STARTING POSITION OF GRID

XS     IS THE SPACE OF GRID IN X DIRECTION.
YS     IS THE SPACE OF GRID IN Y DIRECTION
M      IS THE NUMBER OF DIVISION IN X DIRECTION.
N      IS THE NUMBER OF DIVISIONS IN Y DIRECTION.
```

```
SUBROUTINE MAXPEN(IPEN)

SET PEN TO MAXIMUM PENNUMBER AND RETURNS THIS
```

```
SUBROUTINE NEWPEN(IPEN)

DEFINE A NEW PEN
```

```
SUBROUTINE PLOTS(IDEV,ISIZE,ILUN)

RESET PLOTTING DEVICE
IDEV:
 0=640 X 400    OLIVETTI ATT-6300
 1=640 X 200    IBM HIRES
 2=320 X 200    IBM COLOR
 3=320 X 200    IBM MONOCHROME
 4=720 X 350    HERCULES MONOCHROME
 5=320 X 200    EGA COLOR
 6=640 X 200    EGA COLOR
 7=640 X 350    EGA MONOCHROME
 8=640 X 350    EGA COLOR 4 COLOR
 9=640 X 350    EGA COLOR 16 COLOR
10=640 X 400    ERICSSON
11=640 X 350    IBM MDGA
12=640 X 350    IBM VGA 2 COLOR
13=640 X 350    IBM VGA 64 COLOR

  ADD 100 TO IDEV FOR A PRINT OUTPUT : ILUN=LOGICAL UNIT
IF ISIZE < 0 READ IDEV FROM KEYBOARD
IF ILUN  < 0 READ IDEV FROM FILE 'MONITOR.TYP'
```

```
SUBROUTINE SHOWLINE(IX1,IY1,IX2,IY2)

DRAWS A LINE FROM X1,Y1 TO X2,Y2, IN SCREEN COORDINATES
```

```
SUBROUTINE SYMBOL(XPAGE,YPAGE,HEIGH,IBCD,ANGLE,NCHAR)
```

DRAW A SYMBOL STRING

```
XPAGE,YPAGE      = X,Y COORDINATE          999 LAST X
HEIGHT           = HEIGHT IN CM            999 LAST Y
IBCD             = TEXT STRING
ANGLE            = ANGLE IN DEGREES
+NCHAR           = NO OF CHARACTERS

SPECIAL : SINGLE SYMBOL
IBCD             = INTEGER 0..15 ( ICHAR(0 .. 15) )
-NCHAR           = -1 PEN IS UP WHILE MOVING
                 = -2 PEN IS DOWN WHILE MOVING
SPECIAL : SET MODE
X                = ASPECT RATIO 1=NORMAL ; 999 PREVIOUS
Y                = TALIC 0=VERTICAL ; <>0 SLANT ; 999 PREVIOUS
-HEIGHT          = SET MODE < 0
NCHAR            = CHARACTER SET
                   0 = STANDARD LINE DRAWING     ANY ANGLE
                   1 = DOTS DIRECT WRITE         90 DEGREE ANGLE
                         (GARANTIE ALEEN OP OLIVETTI HIRES MODE)
                   2 = DOTS HIRES 7X14           90 DEGREE ANGLE
                   3 = DOTS LORES 5X7            90 DEGREE ANGLE
```

```
SUBROUTINE VIEWPORT(ILUN,X1,X2,Y1,Y2,IROT)

SET PLOTTING VIEWPORT

ILUN     = LOGICAL UNIT NUMBER  < 0 : ONLY ROTATION
X1       = LEFT  X        0.0 <= X1 <= 1.0
X2       = RIGHT X        0.0 <= X2 <= 1.0
Y1       = BOTTOM Y       0.0 <= Y1 <= 1.0
Y2       = TOP   Y        0.0 <= Y2 <= 1.0
IROT     = NUMBER OF ROTATIONS OF 90 DEGREES BEFORE MAPPING WINDOW
           ON VIEWPORT

VIEWPORTS WORDEN AFGEROND OP EEN RASTER VAN 80 BY 25 VLAKKEN
DIT VANWEGE DE SNELLE BIOS ROUTINES IN AFANUMERIEKE MODE.
```

```
SUBROUTINE WINDOW(X1,X2,Y1,Y2)

SET PLOTTING WINDOW
X1       = LEFT  X
X2       = RIGHT X
Y1       = BOTTOM Y
Y2       = TOP   Y
```

SURFC

The SURFC subroutine produces a 3-dimensional plot of the surface defined in the input array. SURFC scales the X and Z axis over 20 units and the Y axis over units. This means that the plot uses maximal 27.1 units in both X and Y direction on the paper. A hidden line algorithm has been included to remove all lines hidden by the surface.

```
CALL SURFC (ARRAY, IXLEN, IYLEN, NLIN)
```

```
ARRAY = 2-dimensional array (X,Y) containing the Z values.
IXLEN = number of points along the X-axis
IYLEN = number of points along the Y-axis
NLIN  = number of lines drawn over the surface
        If NLIN < 0 no bottom view of the surface is made.
        If IYLEN/(NLIN-1) is not a whole number a linear
        interpolation is done.
```

PRSPEC

The PERSPC subroutine produces a 3-dimensional plot of the surface defined in the input array. A hidden line algorithm has been included to remove all lines hidden by the surface.

```
CALL PRSPEC (Z,IROW,ICOL,NROW,NCOL,IV,R,THETA,PHI,PSIZE)
```

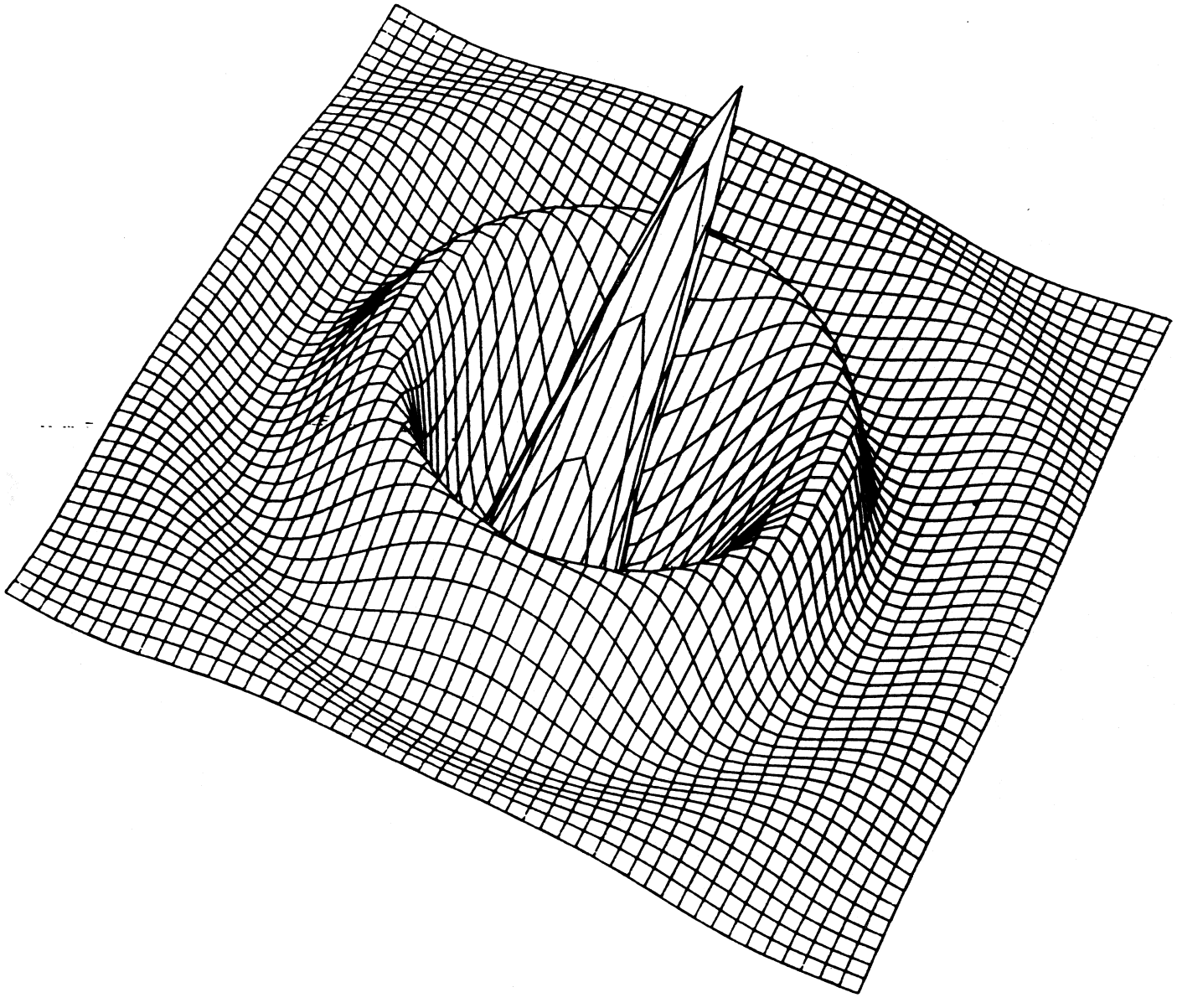
```
Z     = Two dimensional array (NROW,NCOL) containing a
        grid (IROW,ICOL) of surface values
IROW  = Actual X-dimension of the grid
ICOL  = Actual Y-dimension of the grid
NROW  = X-dimension of surface and scratch array
NCOL  = Y-dimension of surface and scratch array
IV    = Two dimension scratch array (NROW,NCOL)
R     = Distance of viewing point (cm)
PHI   = Eulers angle of view point orientation
THETA= Eulers angle of view point orientation
PSIZE= Paper size (cm)
```

```
LINK filename,SYS$LIBRARY:PERSPEC,SYS$LIBRARY:DIPLIB/LIB
```

EXAMPLE OF SURFC ROUTINE

YSTEM PLOT3D 22-DEC-1982 12:13:03

EXAMPLE OF PRSPEC ROUTINE



THREED

Subroutine for perspective pictures (incl contours)

```
CALL THREED(Z,NX,NY,ZLEV,NLEV,X1,Y1,XL,YL,ZBASE,ZMAG,R,
            THETA,PHI,X1PL,Y1PL,XLPL,YLPL,LABPT,LEVFL)
```

```
Z     = Array (1...NX,1...NY) of heights
        (undefined = 10**35)
NX    = No. of grid points in X direction
NY    = No. of grid points in Y direction
ZLEV  = Array of contour levels
NLEV  = No. of contour levels
X1,Y1= Coordinates of Z(1,1) on the plotter
XL,YL= Coordinates of Z(NX,NY) on the plotter
ZBASE= Base level of the figure being plotted.
ZMAG  = Magnification of Z to be used.
R     = Distance to vieuwing point
THETA= Longitude (degrees)  (zero along x axis)
PHI   = Latitude (degrees)
X1PL,
Y1PL  = Lower left hand corner of space on plotter
        for the picture to fit into
XLPL,
YLPL  = Upper right hand corner of space on plotter
        for the picture to fit into
LABPT= Not used
LEVFL= Level indicator. A value of zero indicates
        that contour lines are to be plotted
```

```
LINK  filename,SYS$LIBRARY:3D/LIB,SYS$LIBRARY:DIPLIB/LIB
```

NOTE: THE DATA IN ARRAY Z(I,J) IS MODIFIED BY THIS SUBROUTINE

THREED

Subroutine for perspective pictures

```
CALL THREED(Z,NX,NY,ZLEV,NLEV,X1,Y1,XL,YL,ZBASE,ZMAG,R,
            THETA,PHI,X1PL,Y1PL,XLPL,YLPL,LABPT,LEVFL)
```

This subroutine projects a VOLUME V(x,y,z) on a plane perpendicular to the vector (R,THETA,PHI).

The volume V(x,y,z) is constructed by :

- Surface Z(x,y) to be examined ,
- Baselevel surface ZB(x,y,z) = ZBASE (=constant)
- 4 planes connecting the two surfaces :
  1. plane with constant X=X1
  2. plane with constant Y=Y1
  3. plane with constant X=XL
  4. plane with constant Y=YL

The REFERENCE POINT of the projection is the CENTRE of the baseplane with the coordinates (Xc = (X1+XL)/2 ; Yc = (Y1+YL)/2 ; Zc = ZBASE).

At the end the subroutine THREED scales the projected image of V(x,y,z) to the required size on the plotter (X0PL,X1PL,Y0PL,Y1PL) .

An option is provided to draw CONTOURLINES on the surface Z(x,y) .

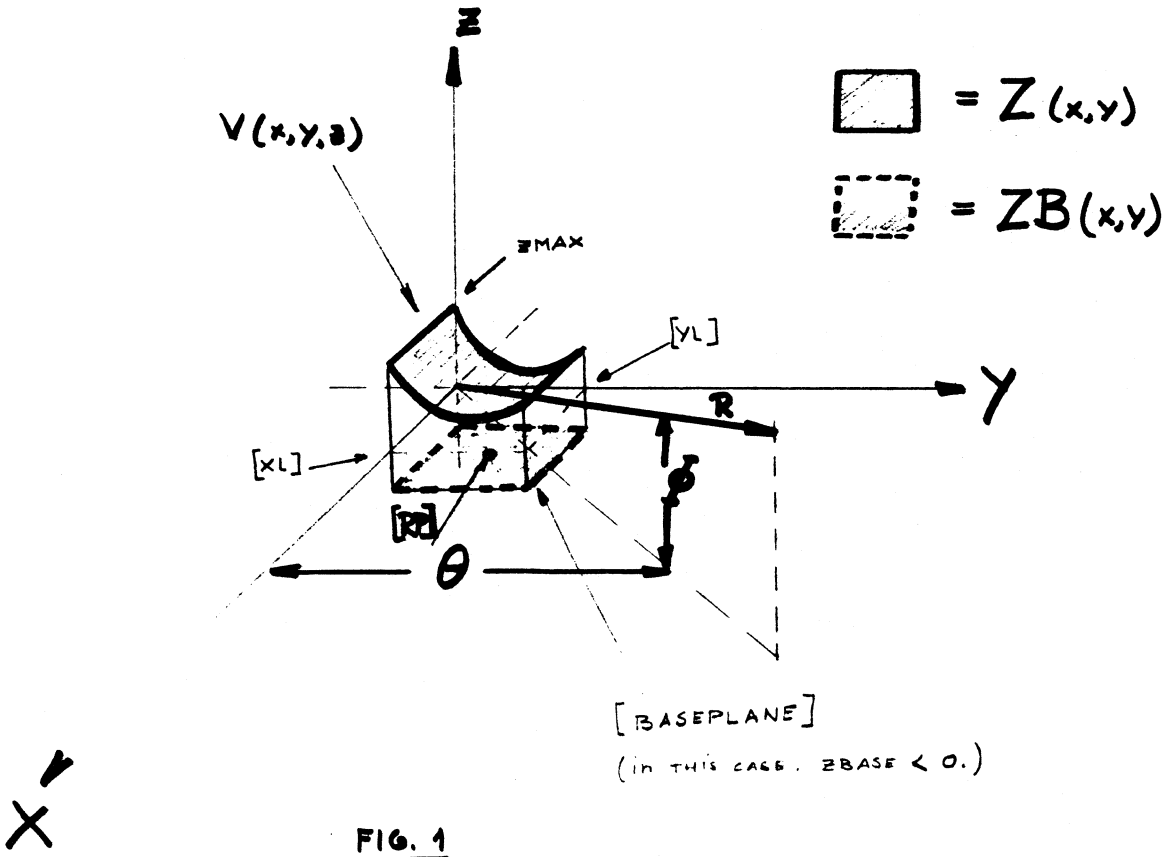


FIG. 1

IPUT PARAMETERS : CALL THREED(Z,NX,NY,ZLEV,NLEV,X1,Y1,XL,YL
ZBASE,ZMAG,R,THETA,PHI,X1PL,Y1PL
XLPL,YLPL,LABPT,LEVFL)

Z	Two dimensional array (= surface Z(x,y)) , containing (NX,NY) datapoints to be plotted as a 3D-plot .
NX,NY	Dimensions of array Z .
ZLEV	Array (NLEV long) containing the Z-values of the contours to be drawn .
NLEV	Number of contourlevels .
X1,Y1	Coordinates of BASEPLANE ZB(x,y) in CM .
XL,YL	Coordinates of BASEPLANE ZB(x,y) in CM .
ZBASE	Z-value of BASEPLANE .

-> (X1,Y1,XL,YL,ZBASE) determine the position of the BASEPLANE and the REFERENCE POINT of the projection : Xc=(XL-X1)/2 , Yc=(YL-Y1)/2 , Zc=ZBASE .
-> (X1,XL) determine the spacing of the lines of Z(x,y) with Y=constant via : DX=(XL-X1)/(NX-1) .
-> (Y1,YL) determine the spacing of the lines of Z(x,y) with X=constant via : DY=(YL-Y1)/(NY-1) .

ZMAG	Magnification of the values of Z(x,y) in the z-direction with respect to the BASEHEIGHT ZBASE : Znew(x,y) = ZMAG * (Z(x,y) - ZBASE)
R	Distance of viewing point . Suggested is to take R a multiple value of (ZMAX-ZMIN)*ZMAG where ZMAX,ZMIN are z-extrema of the volume V(x,y,z) , defined in figure 1 .
THETA PHI	Angles of viewing point . For the orientation of the angles see figure 1 .

-> (R,THETA,PHI) determine the projectionplane , which is perpendicular to this vector .
-> (R) determines the actual size of the projection of V(x,y,z) on the projection plane .

X1PL, Y1PL	Lower left hand corner of the eventual picture on the plotter .
XLPL, YLPL	Upper right hand corner of the eventual picture on the plotter .

-> The projection is automatically scaled down to fit the minimum (X1PL,Y1PL) and maximum (XLPL,YLPL) plotcoordinates .

LABPT	Not used , dummy argument .
LEVFL	Level indicator . LEVFL = 0 indicates that contourlines are drawn .

VION (3P)

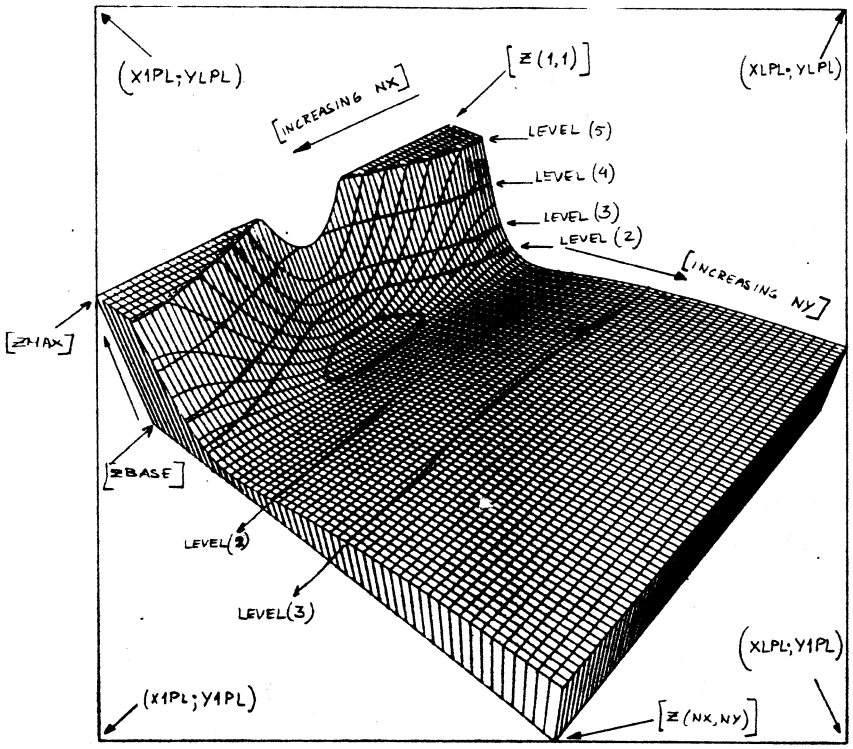
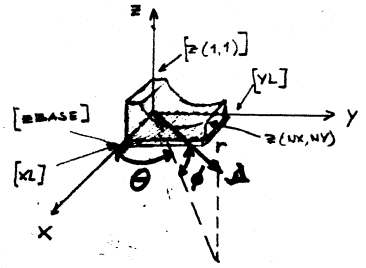


FIG2. : VOORBEELD SUBROUTINE THREED

MIN Z(X,Y)		=	-1.26
MAX Z(X,Y)		=	5.10
ZBASE		=	-1.26
ZMAC		=	1.0
R		=	31.8
THETA	(DEG)	=	40.0
PHI	(DEG)	=	45.0
ZBASE	X1 (CM)	=	0.0
	Y1 (CM)	=	0.0
	XL (CM)	=	18.0
	XL (CM)	=	20.0
PLOT	X1 (CM)	=	0.0
	Y1 (CM)	=	0.0
	XL (CM)	=	20.0
	YL (CM)	=	20.0
NLEV		=	5.
Z LEVEL	(1)	=	-1.00
LEVEL	(2)	=	0.00
LEVEL	(3)	=	1.00
LEVEL	(4)	=	3.00
LEVEL	(5)	=	5.00



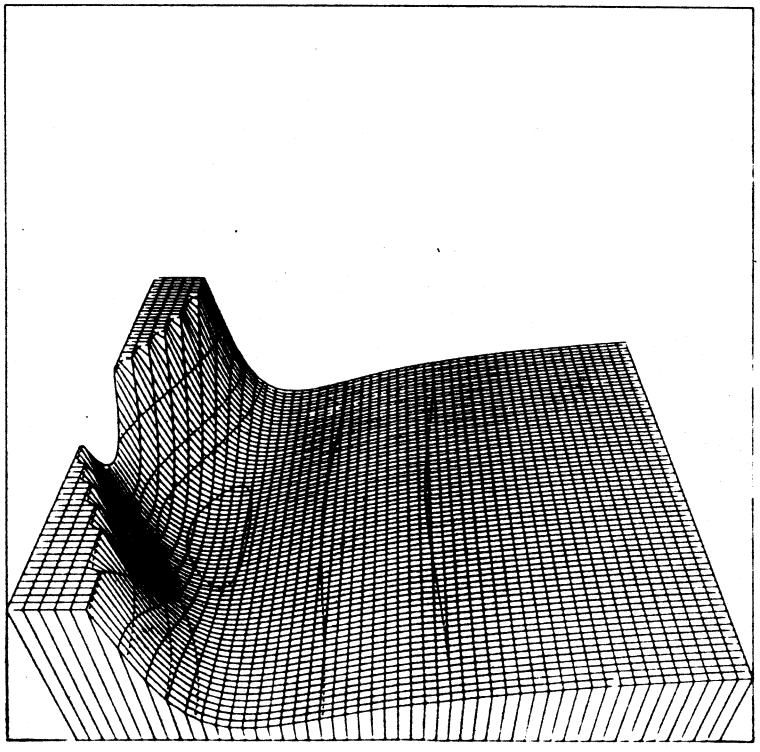
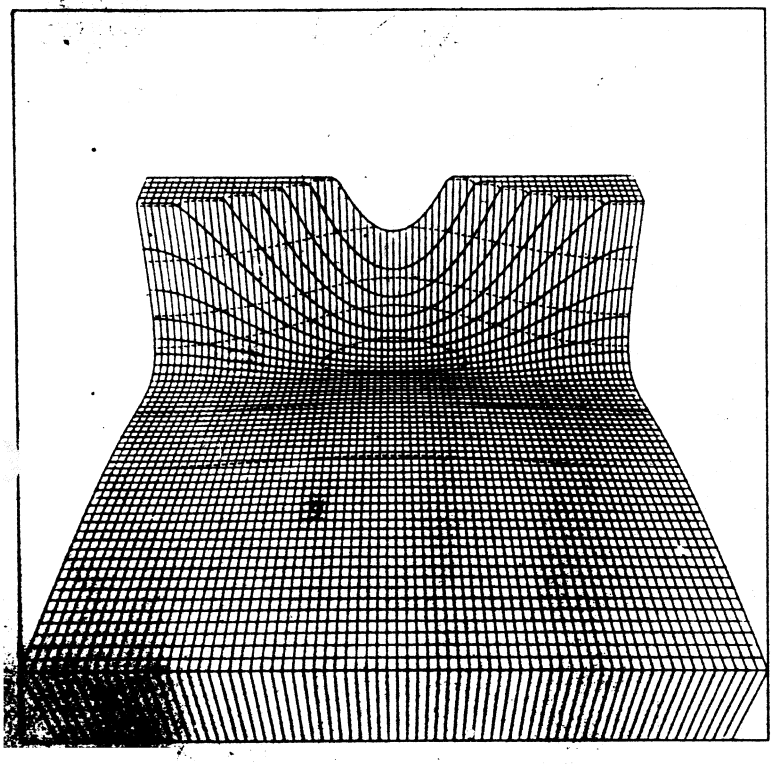


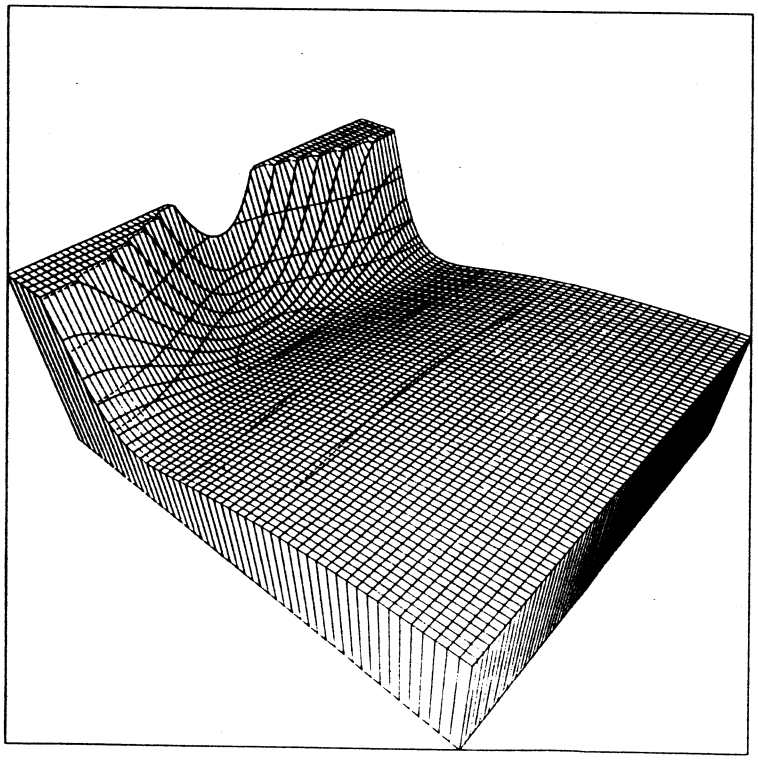
FIG.3 : THETA VERANDERD T.O.V. FIGUUR 2

MIN Z(X,Y)		=	-1.26
MAX Z(X,Y)		=	5.10
ZBASE		=	-1.26
ZMAC		=	1.0
R		=	31.8
THETA	(DEG)	=	0.0
PHI	(DEG)	=	45.0
ZBASE	X1 (CM)	=	0.0
	Y1 (CM)	=	0.0
	XL (CM)	=	18.0
	XL (CM)	=	20.0
PLOT	X1 (CM)	=	0.0
	Y1 (CM)	=	0.0
	XL (CM)	=	20.0
	YL (CM)	=	20.0
NLEV		=	5.
Z LEVEL	(1)	=	-1.00
LEVEL	(2)	=	0.00
LEVEL	(3)	=	1.00
LEVEL	(4)	=	3.00
LEVEL	(5)	=	5.00



MIN Z(X,Y)			= -1.26
MAX Z(X,Y)			= 5.10
ZBASE			= -1.26
ZMAC			= 1.0
R			= 31.8
THETA	(DEG)		= 90.0
PHI	(DEG)		= 45.0
ZBASE	: XI	(CM)	= 0.0
	: YI	(CM)	= 0.0
	: XL	(CM)	= 18.0
	: YL	(CM)	= 20.0
PLOT	: XI	(CM)	= 0.0
	: YI	(CM)	= 0.0
	: XL	(CM)	= 20.0
	: YL	(CM)	= 20.0
NLEV			= 5.
Z	LEVEL	(1)	= -1.00
	LEVEL	(2)	= 0.00
	LEVEL	(3)	= 1.00
	LEVEL	(4)	= 3.00
	LEVEL	(5)	= 5.00

THETA VERANDERD T.O.V. FIG. 2 EN 3



MIN Z(X,Y)			= -1.26
MAX Z(X,Y)			= 5.10
ZBASE			= -3.00
ZMAC			= 1.0
R			= 31.8
THETA	(DEG)		= 40.0
PHI	(DEG)		= 45.0
ZBASE	: XI	(CM)	= 0.0
	: YI	(CM)	= 0.0
	: XL	(CM)	= 18.0
	: YL	(CM)	= 20.0
PLOT	: XI	(CM)	= 0.0
	: YI	(CM)	= 0.0
	: XL	(CM)	= 20.0
	: YL	(CM)	= 20.0
NLEV			= 5.
Z	LEVEL	(1)	= -1.00
	LEVEL	(2)	= 0.00
	LEVEL	(3)	= 1.00
	LEVEL	(4)	= 3.00
	LEVEL	(5)	= 5.00

FIG 5 : ZBASE VERANDERD T.O.V. FIG 2.

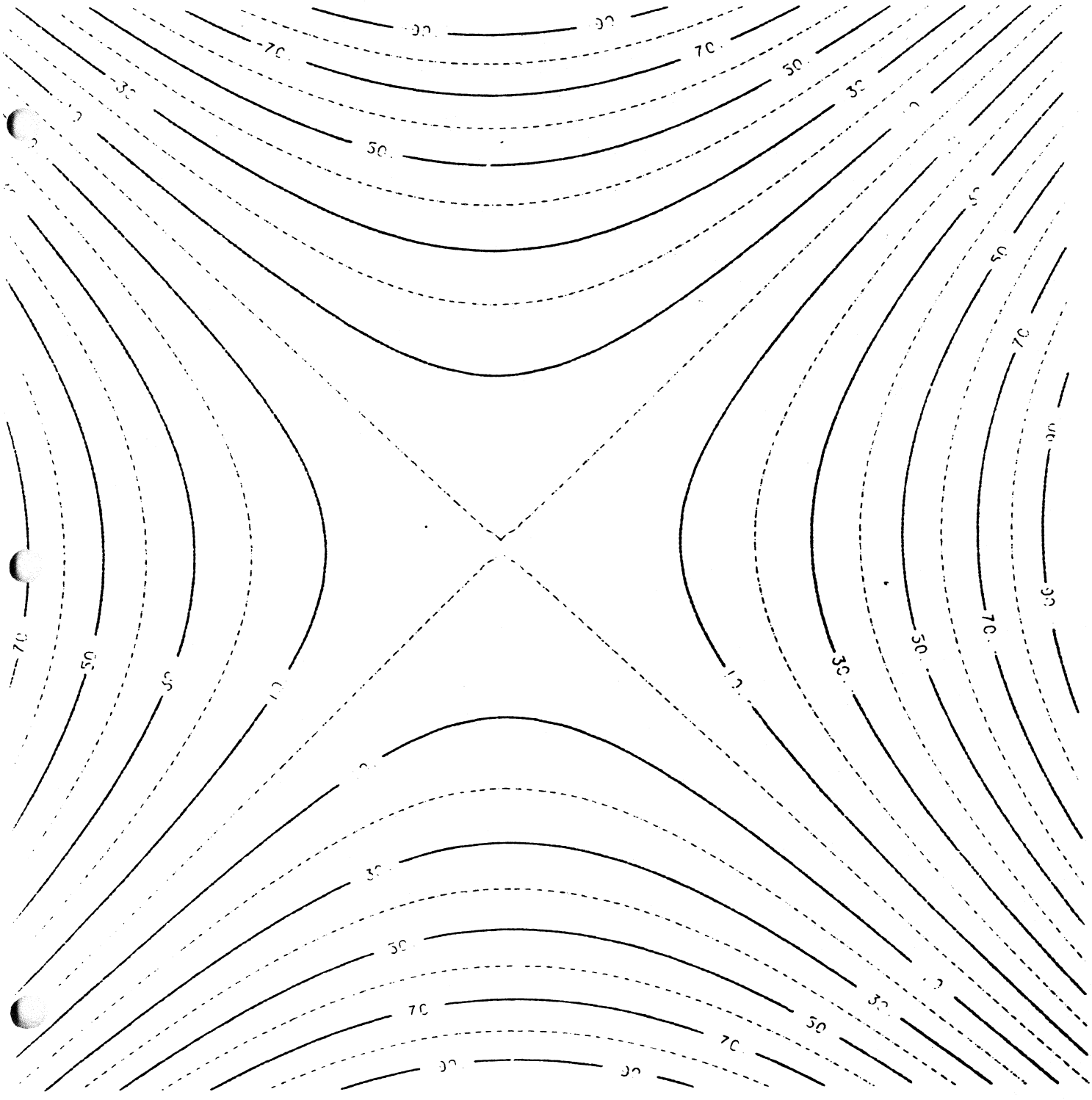
CONSEG

Given heights Z defined over a rectangular grid, contours are drawn at specified levels. There are provisions for labeling the contours.

```
CALL CONSEG(Z,NX,NY,X1,Y1,XL,YL,ZLEV,NDECL,LWGTL,NLEV,HGT,
           NDIV,NARC)
```

```
Z     = Is the input array of heights. Points in
        undefined regions of Z should be set to 10**35
NX,NY= Dimensions of array Z
X1,Y1= Coordinates of Z(1,1) on the plotter
XL,YL= Coordinates of Z(NX,NY) on the plotter
ZLEV  = Is the array of levels to be contoured
NDECL= Gives the number of decimal places in the
        contour labels. -1=no decimal. -2 or less=no label.
LWGTL= Gives the weight of the contour line.
        1 or less=standard line.
        2=heavy line.
        3=dotted line.
NLEV  = Lenght of arrays ZLEV, NDECL, LWGTL.
HGT   = Height of characters of labels
NDIV  = Each grid square is subdivided into NDIV**2
        subsquares using cubic polynomial interpolation.
        NDIV must be a power of 2.
NARC  = 1,2,3 ... 10 .  An arc of narc subsegments replaces
        each line segment of a contour. the arc will match
        slopes with the adjacent arcs. Care should be taken
        here as overlapping of contours is possible when
        narc is used.  NARC=1 has no effect.
```

```
LINK  filename,SYS$LIBRARY:3D/LIB,SYS$LIBRARY:DIPLIB/LIB
```



GETLEV

Subroutine to calculate the contour intervals

```
CALL GETLEV(Z,NX,NY,DZ,ZLEV,NLEV)

Z    = Array (1...NX,1...NY) of heights
       (undefined = 10**35)
NX   = No. of grid points in X direction
NY   = No. of grid points in Y direction
DZ   = Contour interval
ZLEV = Array of contour levels to be calculated
NLEV = No. of contour levels
       If NLEV = 0, DZ and NLEV are determined by this
       routine.


LINK  filename,SYS$LIBRARY:3D/LIB
```

INSIDE

Given a point X,Y and the series XB(k),YB(k) (k=1...NB) defining vertices of a closed polygon. IND is set to 1 if the point is in the polygon and 0 if outside. Each time a new set of bound points is introduced IND should be set to 999 on input. It is best to do a series of Y for a single fixed X. method ... a count is made of the no. of times the boundary cuts the meridian thru (X,Y) south of (X,Y). An odd count indicates the point is inside , even indicates outside.

```
CALL INSIDE (X,Y,XB,YB,NB,IND)


LINK filename,SYS$LIBRARY:3D/LIB
```

ZGRID

Sets up square grid for contouring , given arbitrarily placed data points. LAPLACE interpolation is used. The method used here was lifted directly from notes left by Mr IAN CRAIN. To get smoother results a portion of the beam equation was added to the Laplace equation giving

$$\frac{\partial^2 X}{\partial z^2} + \frac{\partial^2 Y}{\partial z^2} - K\left(\frac{\partial^4 X}{\partial z^4} + \frac{\partial^4 Y}{\partial z^4}\right)$$

K = 0 gives pure Laplace solution.
K = ∞ gives pure spline solution.

```
CALL ZGRID(ZPIJ,KNXT,IMNEW,Z,NX,NY,X1,Y1,DX,DY,XP,YP,ZP,N, CAYIN,NRNG)
```

ZPIJ = Scratch array (1...N) to contain intermediate results.
KNXT = Scratch array (1...N) to contain intermediate results.
IMNEW = Scratch array (1...NY) to contain intermediate results.
Z = 2-d array of heigts to be set up. Points outside the region to be contoured should be initialized to 10**35. The rest should be 0.0.
NX,NY = Max subscripts of Z in X and Y directions .
X1,Y1 = coordinates of Z(1,1)
DX,DY = X and Y increments .
The grid points Z(IX,IY) are calculated by
IX = (XP(I) - X1) /DX + 1.5 and
IY = (YP(I) - Y1) /DY + 1.5
XP,YP,ZP = Arrays giving position and height of each data point.
N = Size of arrays XP,YP and ZP .
CAYIN = Amount of spline equation (between 0 and inf.) (See K in formula above)
NRNG = Grid points more than NRNG grid spaces from the nearest data point are set to undefined.

```
LINK filename,SYS$LIBRARY:3D/LIB
```

SMOOTH

Laplacian smoothing is applied to Z NSM times by means of the operation Z = Z + .25*(AV(ZN,ZS,ZE,ZW)-Z) . Unused points in z should be >= 10**35 .

```
CALL SMOOTH (Z,NX,NY,NSM)

Z      = Array of heights.
NX,NY  = Dimensions of Z
NSM    = Number of passes


LINK filename,SYS$LIBRARY:3D/LIB
```

APPENDIX B

CALCOMP APPLICATIONS SOFTWARE

THREE-D	Plots three dimensional and prespective views of a function of two independent variables. Will automatically grid and smooth data, draw and annotate axes.
GPCP	A General Purpose Contouring Program to plot contour maps under control of a variety of options.
FLOWGEN	A program which automatically produces and plots a flowchart of any ANS FORTRAN IV program directly from the source deck.
AUTONET	Two versions of AUTONET enable users to plot the output from various levels of network scheduling programs e.g., CPM, PERT, PMS and PCS. Autonet enables users to generate Critical Path Method (CPM) linear network charts from the output of any CPM computer program which supplies the necessary I-node/J-node data.
DATAGRAPH	Produces management information charts and graphs directly from user data on cards, tape or disc.
CONTOUR	A basic contouring package for small machines not holding or needing the flexibility of GPCP.
AUTOGANT	An Automatic Gantt Bar Chart Display Program useful in conjunction with AUTONET.

APPENDIX C

FUNCTIONAL SOFTWARE

General category

CIRCLE	Draws a circle or spiral.
DASHL	Draws dashed lines connecting a series of data points.
DASHP	Draws a dashed line to a specified point.
ELIPS	Draws an elipse or elliptical arc.
FIT	Draws a curve through three points.
GRID	Draws a linear grid.
POLY	Draws an equilateral polygon.
RECT	Draws a rectangle.

Drafting Category

AROHD	Draws various types of arrowheads.
ARROW	Draws a line terminated with an arrowhead through a set of data points.
CNTRL	Draws a "center line" through a set of data points.
DIMEN	Draws annotated dimension lines with extension lines.
LABEL	Draws annotation centered between two points with control over symbol placement.

Business Category

AXISB	Draws an axis with "business" annotation.
AXISC	Draws an axis with calendar month annotation.
BAR	Draws bars, for bar graph plotting.
LBAXS	Draws a logarithmic axis with business annotation.
LGLIN	Plots data in either log-log or semi-log mode.
SHADE	Draws shading between lines.
SCALG	Performs scaling for logarithmic plotting.

Scientific Category

CURVX	Plots a function of X over a given range.
CURVY	Plots a function of Y over a given range.
FLINE	Draws a smooth curve through a set of data points.
LGAXS	Draws a logarithmic axis with annotation.
LGLIN	Plots data in either log-log or semi-log mode.
POLAR	Plots data points using polar coordinates.
SCALG	Performs scaling for logarithmic plotting.
SMOOT	Draws a smooth curve through sequential data points.

Miscellaneous Category

CVPLOT	Polynomial Curve Fitting Routine, accompanied by a driver to allow use of a complete program, which determines and plots a polynomial of given degree which best fits the data points. The least squares approximation technique is used. Data points are also plotted.
FLOCT	Flowchart production program which plots and annotates a flowchart defined by input data cards.
FORGN	Tape and Card Forms Generator which draws the blank form with headings, for tape formats or card forms, as defined by input data cards.
SPECIAL SYMBOL SETS	Drafting Letter Set, Block Letter Symbol Set, Upper Case/Lower Case Symbol Set.
DIGITIZER SUPPORT SOFTWARE	Processes input from CalComp Digitizer using a special menu.
WORLD MAP	A set of subroutines used to plot and annotate various map projects.

CONTENTS

ILLUSTRATIONS

SECTION 1
INTRODUCTION

GENERAL

This user's manual describes the calling sequences and arguments for the Scientific Applications category of CalComp's Functional Software Library. These eight standard FORTRAN subroutines generate calls to CalComp Host Computer Basic Software (HCBS).

Each subroutine description includes plotted-output samples which show the effects of using various argument values. The argument names used in the calling sequences are arbitrary and need not be used. They have been chosen to illustrate the use of the argument and its particular type, i.e., if the initial of the name is I - N, the argument is an Integer type; otherwise, it is a Real type.

Any other Functional subroutines called by the subroutines described herein are supplied with the category package, but are not intended to be called independently.

FUNCTIONAL SUBROUTINES

The subroutine descriptions follow composite sample plots. Figures 1-1 and 1-2 illustrate the usage of all the subroutines.

CURVX – plots a function of X over a given range

CURVY – plots a function of Y over a given range

FLINE – draws a smooth curve through a set of data points

LGAXS – draws a logarithmic axis with annotation

LGLIN – Plots data either in log-log or in semi-log mode

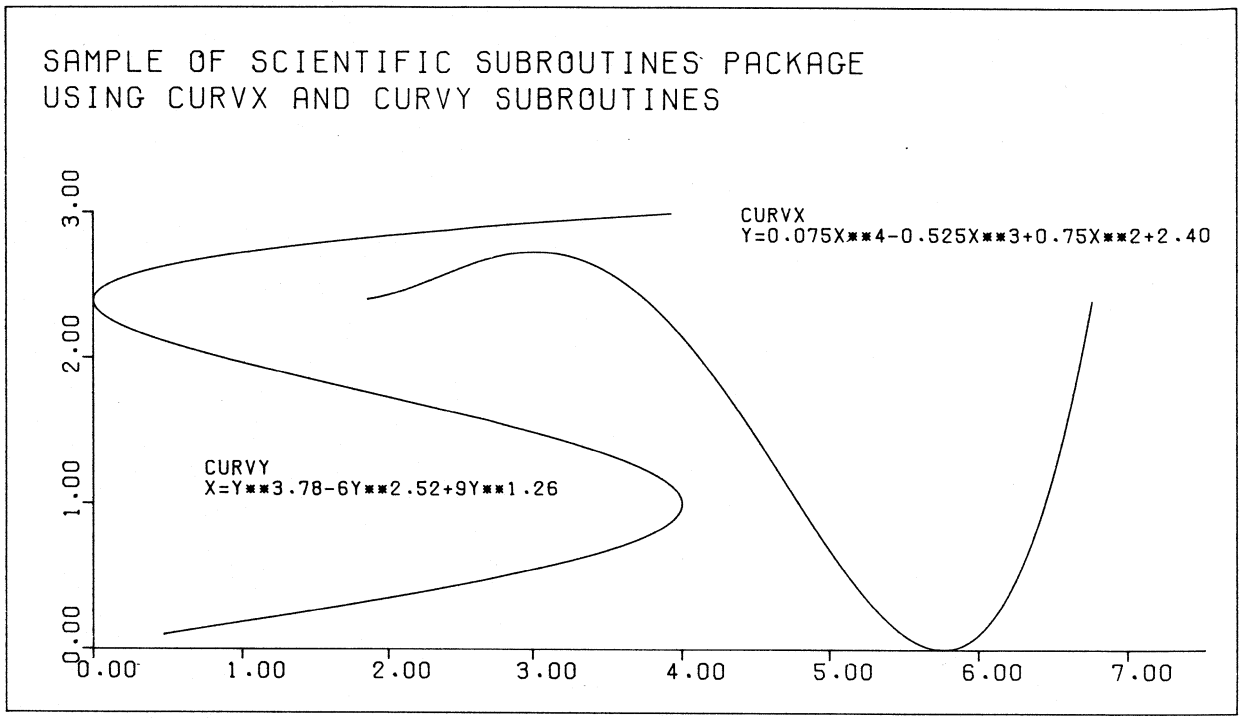
POLAR – plots data points using polar coordinates

SCALG – performs scaling for logarithmic plotting

SMOOT – draws a smooth curve through sequential data points

NOTE

This manual applies to both inch and metric measurement. Metric parameter values are shown in parentheses in all illustrations. Metric or inch measurement is dependent upon the specific Host Computer Basic Software being used.



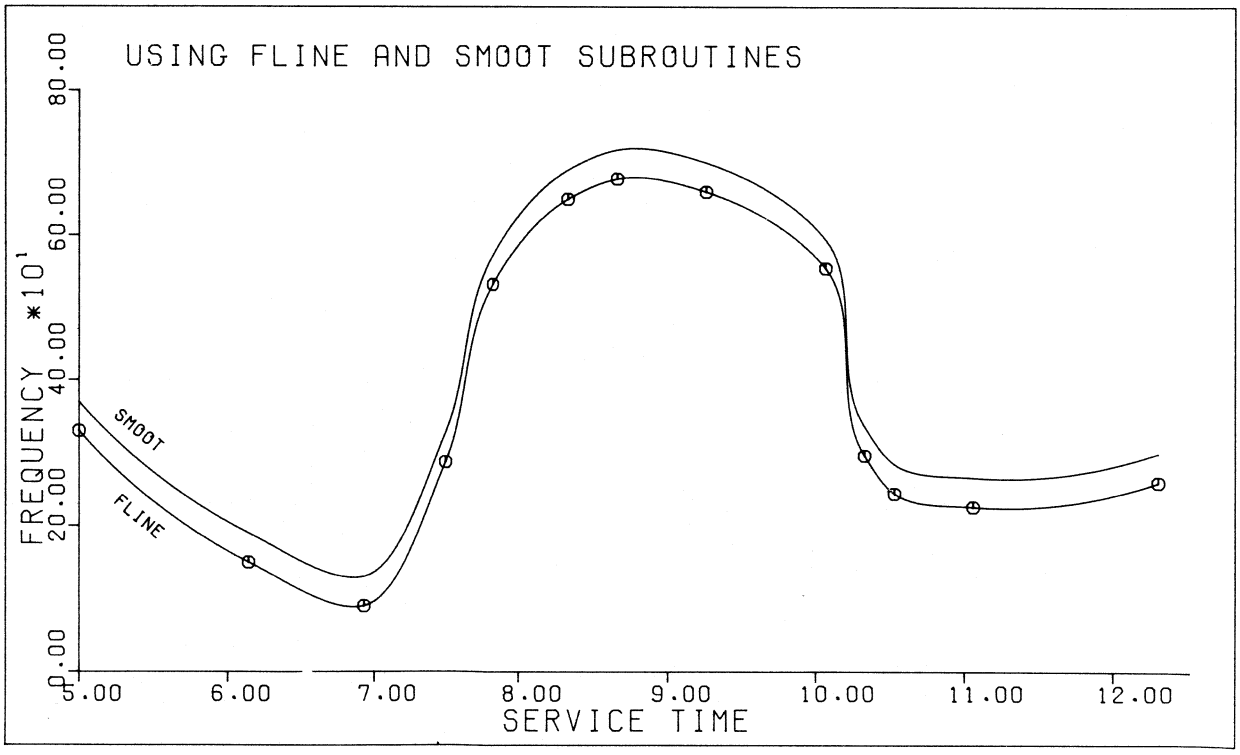


Figure 1-1. Sample of CURVX, CURVY, FLINE, and SMOOT Subroutines

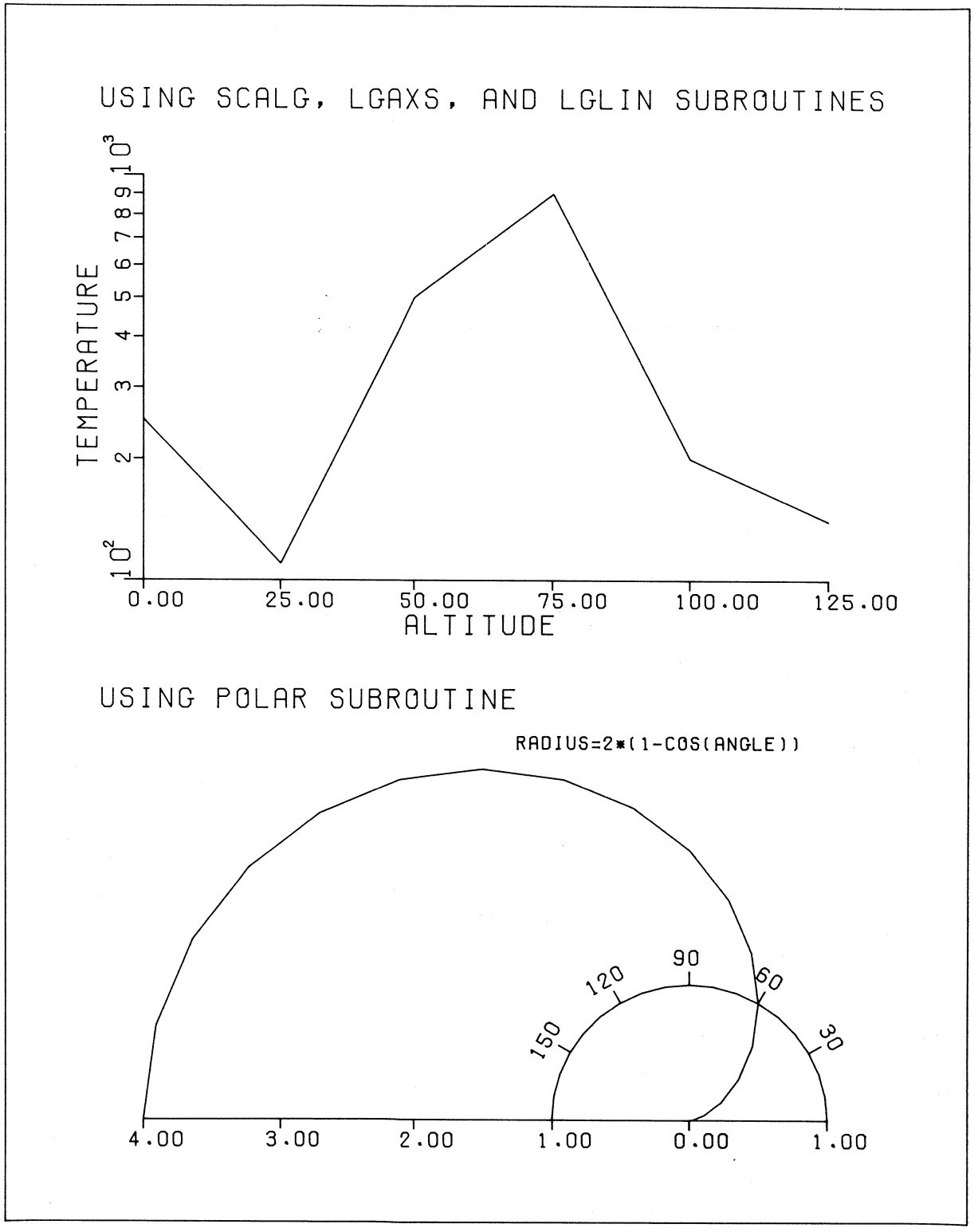
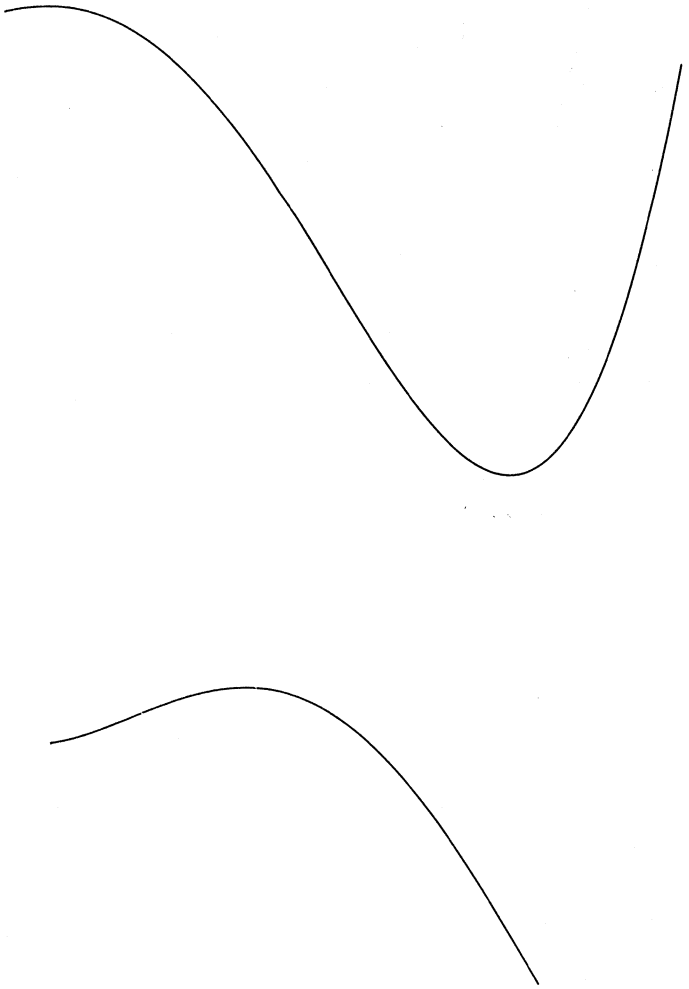


Figure 1-2. Sample of SCALG, LGAXS, LGLIN, and POLAR Subroutines

(XO,XF,COEFF1,EXP1,COEFF2,
EXP2,COEFF3,EXP3,COEFF4,EXP4)

AND XF



SECTION 2
CURVX SUBROUTINE

GENERAL

CURVX is a FORTRAN subroutine which plots a function of X

CALLING SEQUENCE

CALL CURVX (XO,XF,COEFF1,EXP1,COEFF2,EXP2,COEFF

XO,XF	are the starting and ending value meters).
COEFF1,COEFF2, COEFF3,COEFF4	are the coefficients of X in the po function to be plotted.
EXP1,EXP2, EXP3,EXP4	are the exponents of X in the pol function to be plotted.

See Figure 2-1 for examples of parameter usage.

COMMENTS

The polynomial that defines the function to be plotted is:

$$Y=COEFF1 * X^{EXP1}+COEFF2 * X^{EXP2}+COEFF3 * X^{EXP3}+C$$

for values of X from XO to XF, where $\Delta X = 0.01$ (0.0254) assumed to be inches (centimeters) and plotting is done at a required must be performed before calling this subroutine.

CURVX can process only positive, nonzero values of X.

OTHER FUNCTIONAL SUBROUTINES USED

None.

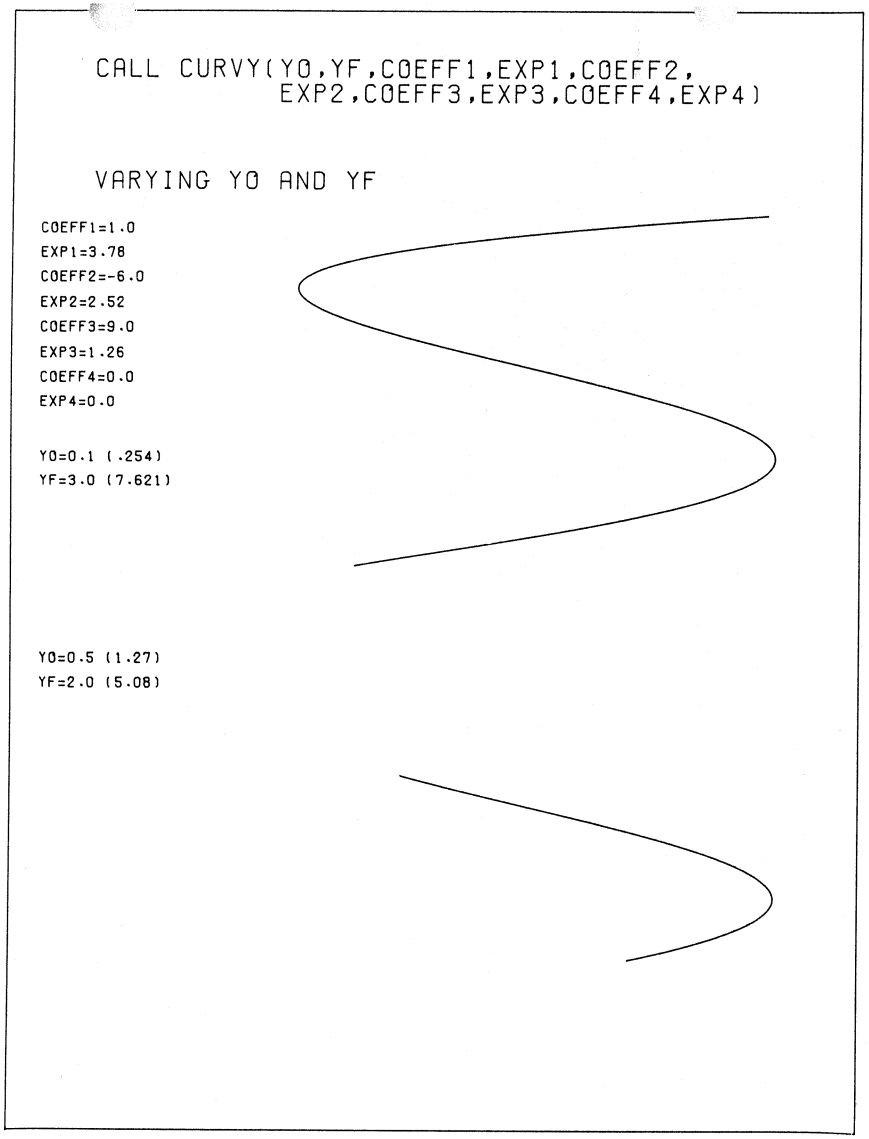


Figure 3-1. Sample of CURVY Subroutine

SECTION 3
CURVY SUBROUTINE

GENERAL

CURVY is a FORTRAN subroutine which plots a function of Y over a given range.

CALLING SEQUENCE

CALL CURVY (YO,YF,COEFF1,EXP1,COEFF2,EXP2,COEFF3,EXP3,COEFF4,EXP4)

YO,YF	are the starting and ending values of Y in inches (centimeters).
COEFF1,COEFF2, COEFF3,COEFF4	are the coefficients of Y in the polynomial that defines the function to be plotted.
EXP1,EXP2, EXP3,EXP4	are the exponents of Y in the polynomial that defines the function to be plotted.

See Figure 3-1 for examples of parameter usage.

COMMENTS

The polynomial that defines the function to be plotted is:

$$X=COEFF1 * Y^{EXP1}+COEFF2 * Y^{EXP2}+COEFF3 * Y^{EXP3}+COEFF4 * Y^{EXP4}$$

for values of Y from YO to YF, where $\Delta Y = 0.01$ (0.0254). Since values of Y are assumed to be inches (centimeters) and plotting is done at a scale of 1.0, any scaling required must be performed before calling this subroutine.

CURVY can process only positive, nonzero values of Y.

OTHER FUNCTIONAL SUBROUTINES USED

None.

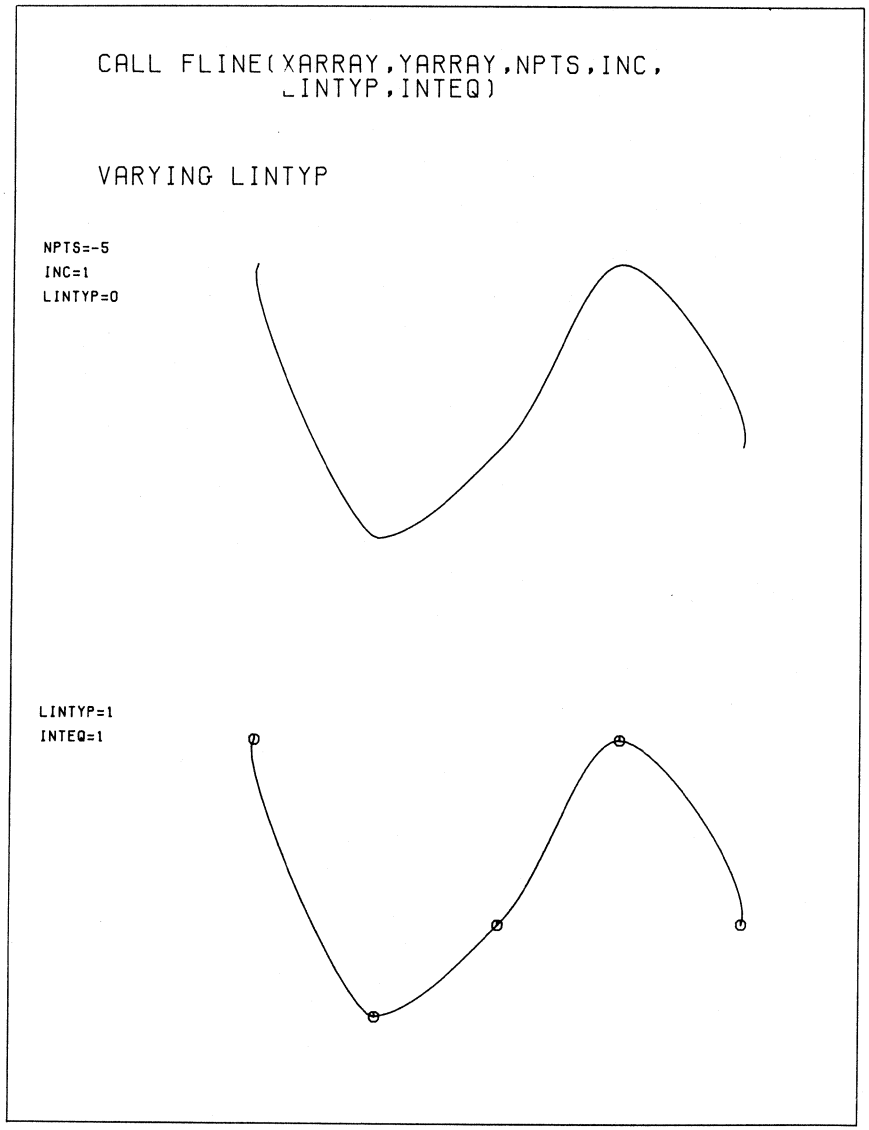


Figure 4-1. Sample of FLINE Subroutine

SECTION 4
FLINE SUBROUTINE

GENERAL

FLINE plots data points from an array. Control is provided for annotating the data points with centered symbols and/or drawing straight lines or smooth curves through each successive data point.

CALLING SEQUENCE

CALL FLINE (XARRAY, YARRAY, NPTS, INC, LINTYP, INTEQ)

XARRAY — is the name of the array containing the variables to be plotted as the abscissas.

YARRAY — is the name of the array containing the variables to be plotted as the ordinates.

The adjusted minimum and the adjusted delta must be stored in the X and Y data arrays. Refer to COMMENTS.

NPTS — is the number of data points to be plotted.

If NPTS > 0, a straight line is drawn between the points.

If NPTS < 0, a smooth curve is created using a modified spline-fitting technique and drawn between points.

INC — is the increment between array elements (normally, INC = 1). INC may be greater than 1 if mixed or multi-dimensioned arrays are used and every INCth value is plotted.

LINTYP — is used to control the type of graph produced:

If LINTYP = 0 a line plot is produced between successive data points.

= 1 a line plot is produced, with a symbol at each data point.

= n a line plot is produced, with a symbol at every nth data point.

= -n connecting lines are not plotted between data points; a symbol appears at every nth data point.

CALLING SEQUENCE (cont)

INTEQ	is the integer equivalent used to specify the symbol to be plotted at a data point. For possible values of INTEQ, refer to the description of the special call to SYMBOL in Programming CalComp Electromechanical Plotters (Manual 1006).

COMMENTS

FLINE is similar to LINE, with the additional capability of drawing a smooth curve connecting data points.

Figure 4-1 illustrates FLINE plotted output.

The arrays XARRAY and YARRAY must be dimensioned at least to (NPTS * INC + INC + 1) so as to contain the adjusted minimum value (FIRSTV), and the adjusted delta value (DELTAV) to be used on the respective axis annotation.

For the X-array, the adjusted minimum is stored in XARRAY (NPTS * INC + 1) and the adjusted delta is in XARRAY (NPTS * INC + INC + 1). For the Y-array, the minimum is in YARRAY (NPTS * INC + 1), and the delta is in YARRAY (NPTS * INC + INC + 1).

If either the XARRAY or YARRAY are to be plotted without translation at a scale factor of 1.0, store 0.0 in FIRSTV and 1.0 in DELTAV.

OTHER FUNCTIONAL SUBROUTINES USED

REFLX
FIT4

SECTION 5
LGAXS SUBROUTINE

GENERAL

LGAXS is a FORTRAN subroutine which draws a logarithmic axis with annotation in powers of ten and a title.

CALLING SEQUENCE

CALL LGAXS (XPAGE,YPAGE,IBCD,NCHAR,AXLEN,ANGLE,FIRSTV,DELTAV)

XPAGE, YPAGE	are the coordinates, in inches (centimeters) of the axis starting point.
IBCD	is the alphameric array data to be used as the axis title (may be one or more characters).
NCHAR	is the number of characters in the axis title. If NCHAR is: negative, the axis, title and coordinate labeling are placed on the clockwise side of the axis; positive, the axis, title and coordinate labeling are placed on the counterclockwise side of the axis.
AXLEN	is the length of the axis, in inches (centimeters).
ANGLE	is the angle, in degrees, at which the axis is to be drawn. The axis is rotated positive counterclockwise from the horizontal line extending to the right from XPAGE, YPAGE.
FIRSTV	is the value of coordinate labeling at the beginning of the axis.
DELTAV	is the number of log cycles per inch (centimeters), or the reciprocal of the length of one cycle in inches (centimeters).

See Figure 5-1 for examples of parameter usage.

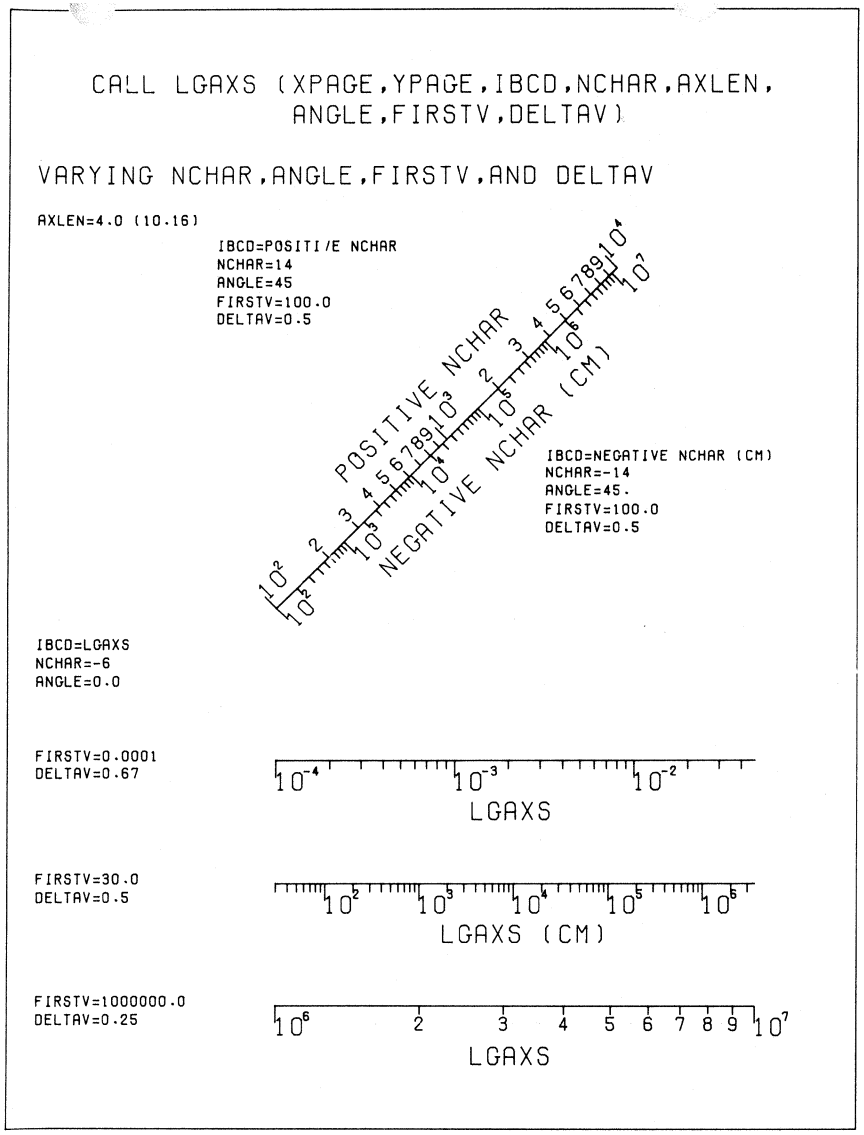


Figure 5-1. Sample of LGAXS Subroutine

COMMENTS

A tick mark is placed on the axis for each power of ten and for each of the nine intermediate integer values.

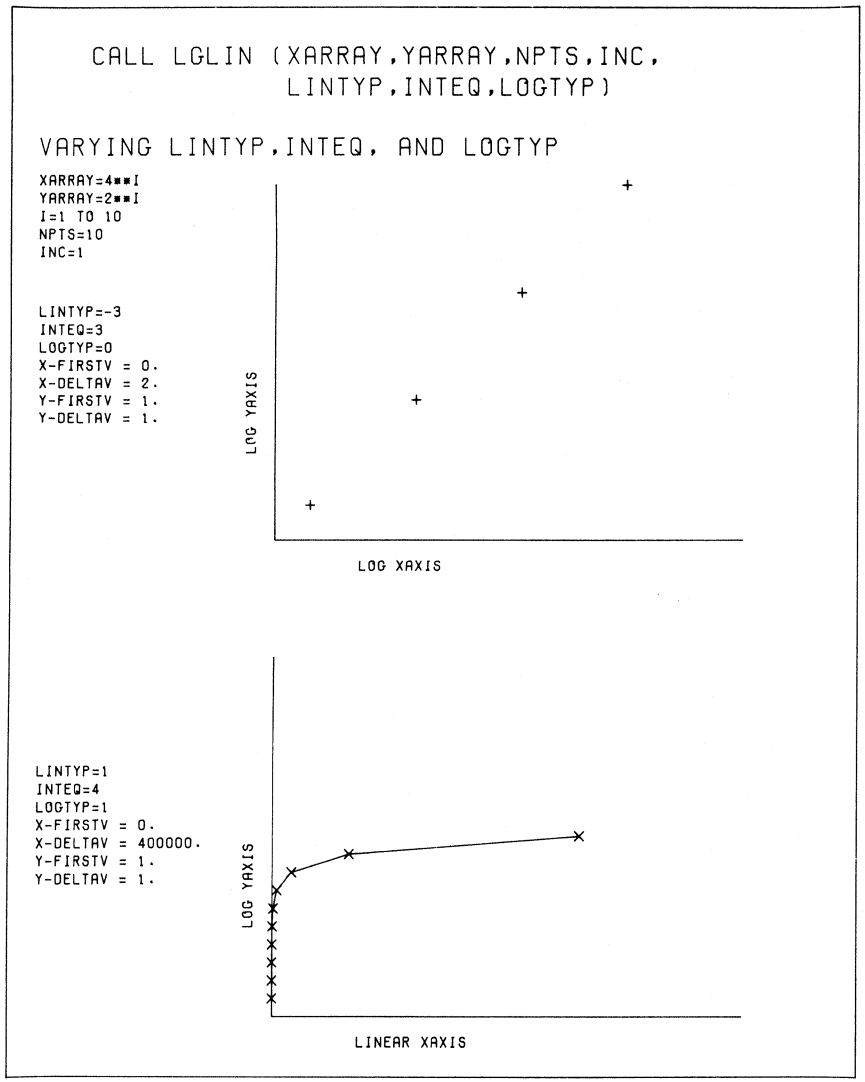
Annotation is placed at the tick marks as follows:

- If a cycle is not less than two inches (centimeters) long, the integer tick marks are annotated.
- The power-of-ten tick marks are annotated in the form 10^N.
- Controlled by AXLEN, the axis line may not end at a tick mark.

The SCALG subroutine may be used for determining FIRSTV and DELTAV.

OTHER FUNCTIONAL SUBROUTINES USED

None.



Figures 6-1. Sample of LGLIN Subroutine

SECTION 6
LGLIN SUBROUTINE

GENERAL

LGLIN is a FORTRAN subroutine used to plot data in either log-log or semi-log mode.

CALLING SEQUENCE

CALL LGLIN (XARRAY,YARRAY,NPTS,INC,LINTYP,INTEQ,LOGTYP)

XARRAY, YARRAY — are the arrays containing the variables to be plotted as abscissa and ordinate, respectively. They are either logarithmic or linear, depending on the value of LOGTYP.

NPTS — is the number of points to be plotted.

INC — is the increment between array elements (normally, INC = 1). INC may be greater than 1 if mixed or multi-dimensioned arrays are used and only every INCth value is to be plotted.

LINTYP — is used to control the type of graph produced:

If LINTYP = 0 — a line is plotted between successive data points and no symbol is produced.

= 1 — a line plot is produced, with a symbol at each data point.

= n — a line plot is produced, with a symbol at every nth data point, starting with the first point.
EXAMPLE: LINTYP = 3, a symbol is plotted at every third point.

= -n — connecting lines are not plotted between data points. A symbol appears at every nth data point, starting with the first point.

INTEQ — is the integer equivalent used to specify the symbol to be plotted at a data point. INTEQ has no effect when LINTYP = 0.

For possible values of INTEQ, refer to Programming CalComp Electromechanical Plotters (Manual 1006) in the description of the special call to SYMBOL.

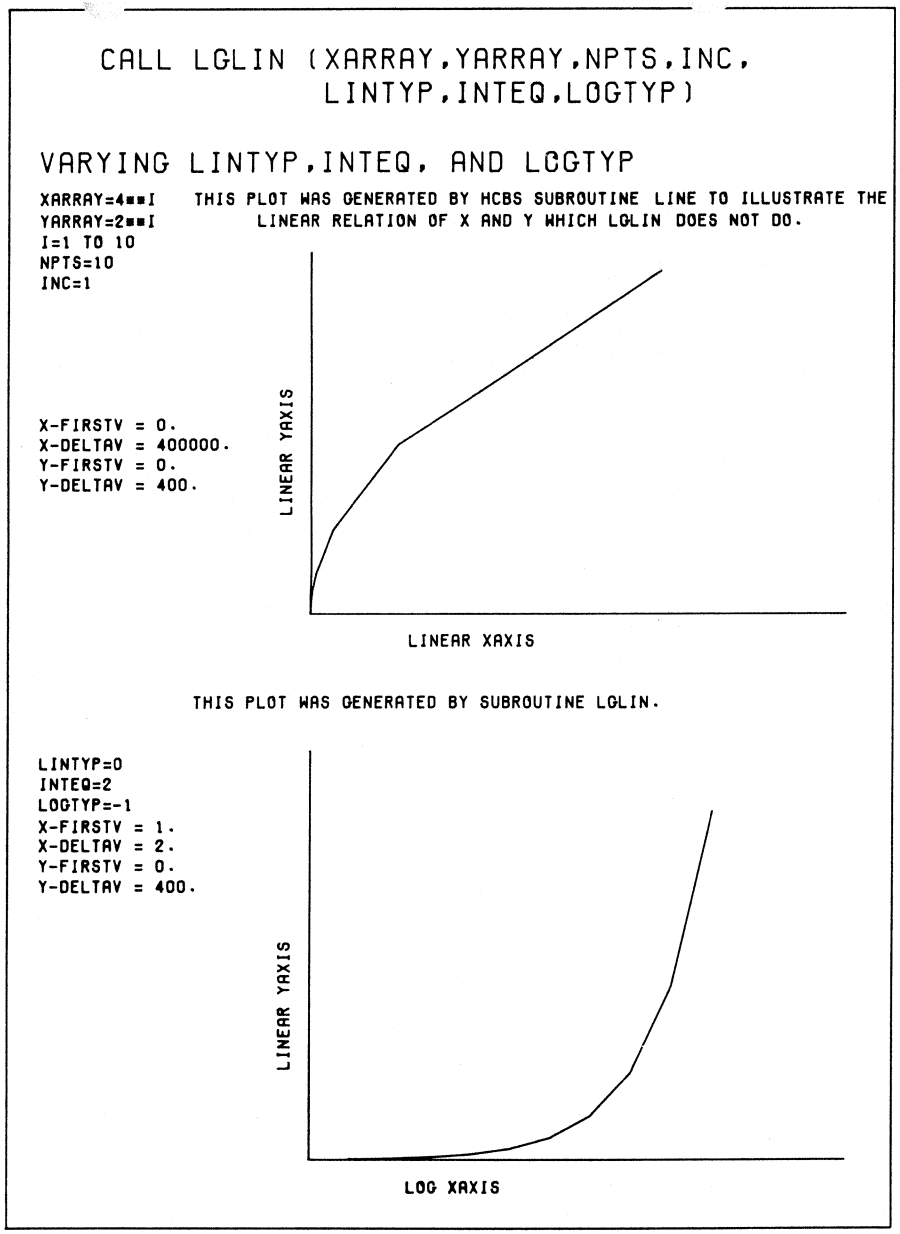


Figure 6-2. Sample of LGLIN Subroutine

CALLING SEQUENCE (cont)

LOGTYP is a code specifying the type of plot.

IF LOGTYP = -1 a plot which is logarithmic in X and linear in Y is produced.

= 0 a plot which is logarithmic in X and Y is produced.

= +1 a plot which is linear in X and logarithmic in Y is produced.

See Figures 6-1 and 6-2 for examples of parameter usage.

COMMENTS

The arrays XARRAY and YARRAY must be dimensioned at least to (NPTS * INC + INC + 1) so as to contain the adjusted minimum value (FIRSTV) and the adjusted delta value (DELTAV) to be used on the respective axis annotation. For the X-array, the adjusted minimum is stored in XARRAY (NPTS * INC + 1) and the adjusted delta is in XARRAY (NPTS * INC + INC +1). For the Y-array, the minimum is in YARRAY (NPTS * INC + 1), and the delta is in YARRAY (NPTS * INC + INC +1).

Depending on the value of LOGTYP, FIRSTV and DELTAV may be provided by the SCALE subroutine for a linear axis, or the SCALG subroutine for a logarithmic axis.

If either the XARRAY or YARRAY are to be plotted without translation at a scale factor of 1.0, store 0.0 in FIRSTV and 1.0 in DELTAV.

Unless LINTYP is negative, a line is drawn connecting sequential points. Movement is optimized so that plotting may either begin at the first point and progress forward or begin at the last point and progress backward in the arrays.

OTHER FUNCTIONAL SUBROUTINES USED

None.

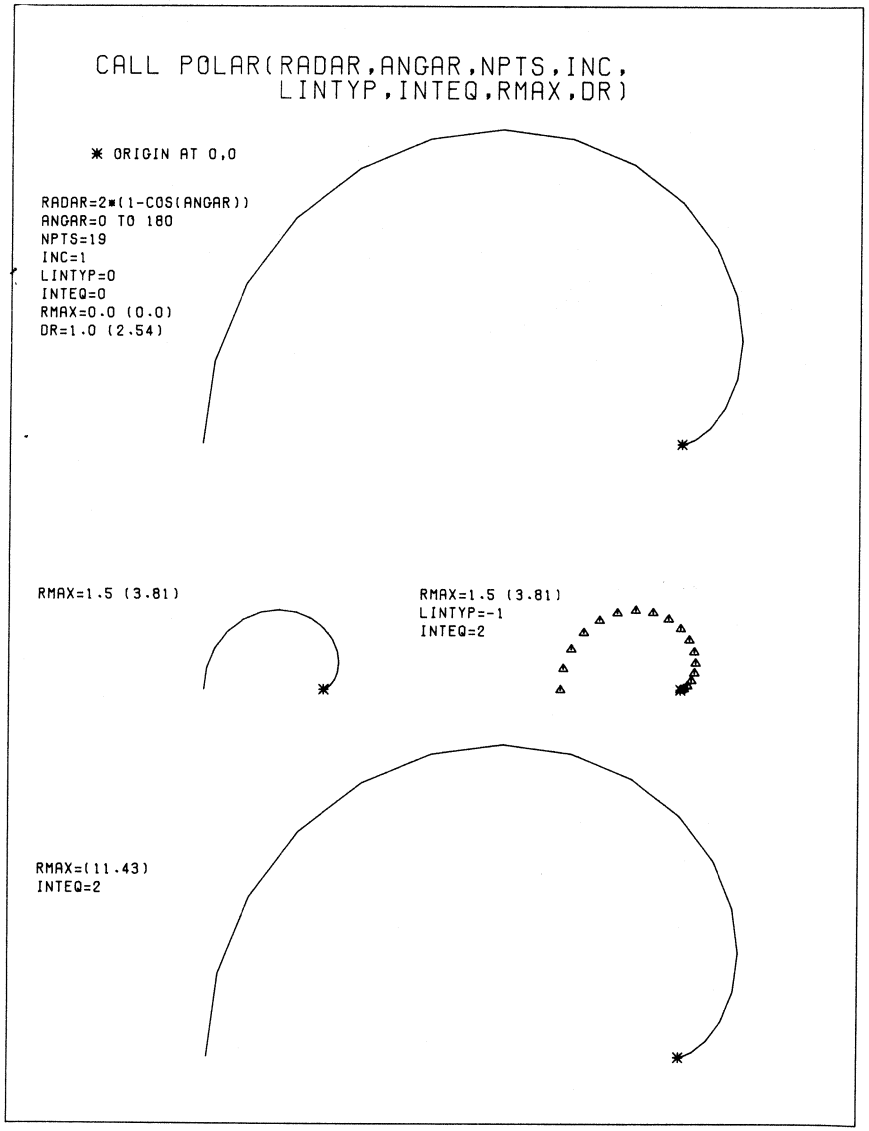


Figure 7-1. Sample of POLAR Subroutine

SECTION 7
POLAR SUBROUTINE

GENERAL

POLAR is a FORTRAN subroutine which scales and plots a radial variable of any magnitude against an angular variable (angle in radians) as polar coordinates. POLAR produces either a line plot (with lines connecting data points) or a point plot, centered at (0.,0.).

CALLING SEQUENCE

CALL POLAR (RADAR, ANGAR, NPTS, INC, LINTYP, INTEQ, RMAX, DR)

RADAR — is the name of the array containing the radial values.

ANGAR — is the name of the array containing the angular values (radians).

NPTS — is the number of data points to be plotted.

INC — is the increment between array elements (normally, INC = 1). INC may be greater than 1 if mixed or multi-dimensioned arrays are used and only every INCth value is to be plotted.

LINTYP — is used to control the type of graph produced:

IF LINTYP = 0 a line plot is produced between successive data points, and no symbol is produced.

= 1 a line plot is produced, with a symbol at each data point.

= n a line plot is produced, with a symbol at every nth data point.

= -n connecting lines are not plotted between data points; a symbol appears at every nth data point.

INTEQ — is the integer equivalent used to specify the symbol to be plotted at a data point. For possible values of INTEQ, see the description of the special call to SYMBOL in Programming CalComp Electromechanical Plotters (Manual 1006).

INTEQ has no effect when LINTYP = 0.

CALLING SEQUENCE (cont)

RMAX — is the maximum radius for the plotting area, in page inches (centimeters). If RMAX $\leq$ 0., DR is used as a scale factor.

DR — is the scale factor. If RMAX $>$ 0., DR is computed by the POLAR subroutine; if RMAX $\leq$ 0., DR must contain the scale factor. DR is expressed in units of data per page inch (centimeter).

See Figure 7-1 for examples of parameter usage.

COMMENTS

Angles are measured in radians positive counterclockwise around (0.,0.), with zero being in the +X direction.

Radial values are measured from (0.,0.), with negative values being plotted radially opposite from positive values.

OTHER FUNCTIONAL SUBROUTINES USED

None.

SECTION 8
SCALG SUBROUTINE

GENERAL

SCALG is a FORTRAN subroutine used to determine scale factors of the data array to be plotted on a logarithmic scale. The scale factors are those used by subroutines LGLIN and LBAXS.

CALLING SEQUENCE

CALL SCALG (ARRAY, AXLEN, NPTS, INC)

ARRAY — is the array containing data to be scaled. As output, an adjusted minimum value is stored in ARRAY (NPTS * INC + 1), and a delta value, log cycles per inch (centimeters) is stored in ARRAY (NPTS * INC + INC +1). ARRAY must be dimensioned at least (NPTS * INC + INC +1) to provide storage for these scale factors.

AXLEN — is the maximum length over which data is to be plotted, in inches (centimeters).

NPTS — is the number of ARRAY values to be scaled.

INC — is the increment between array elements (normally, INC = 1). INC may be greater than 1 if mixed or multidimensioned arrays are used and only every INCth value is to be plotted.

COMMENTS

Every INCth element of ARRAY is selected, e.g., a, b, c, d, and e. n is defined as the largest integer such that:

$$10^n \leqslant \min \{a, b, c, d, e\}$$

m is defined as the smallest integer such that:

$$10^m \geqslant \max \{a, b, c, d, e\}$$

The adjusted minimum is given as:

$$\text{FIRSTV} = \text{ARRAY (NPTS * INC + 1)} = 10^n$$

The adjusted delta is given as:

$$\text{DELTAV} = \text{ARRAY (NPTS * INC + INC + 1)} = \frac{m-n}{\text{AXLEN}}$$

OTHER FUNCTIONAL SUBROUTINES USED

None.

EXAMPLES

A. For the following array of values:

```
ARRAY(1) =  1500.0
ARRAY(2) =  3000.0
ARRAY(3) =  2500.0
ARRAY(4) =   300.0
```

and the following argument values:

```
AXLEN  = 2.0
NPTS   = 4
INC    = 1
```

the adjusted minimum (FIRSTV) and delta value (DELTAV) stored by SCALG are:

```
FIRSTV:      ARRAY(5) =  100.0
DELTAV:      ARRAY(6) =    1.0
```

B. If the value of AXLEN in Example A were changed to 4.0, the resultant values stored would be:

```
FIRSTV:      ARRAY(5) =  100.0
DELTAV:      ARRAY(6) =    0.5
```

C. For the following array of values:

```
 ARRAY(1) =   1.2*
 ARRAY(2) = 100.0
 ARRAY(3) =   2.3*
 ARRAY(4) =  88.0
 ARRAY(5) =   1.8*
 ARRAY(6) =   0.0
 ARRAY(7) =   0.8*
 ARRAY(8) =  20.0
 ARRAY(9) =   0.7*
ARRAY(10) =  10.0
```

and the following argument values:

```
AXLEN  =   10.0
NPTS   =    5
INC    =    2
```

the adjusted minimum (FIRSTV) and delta value (DELTAV) are determined from the value of the asterisked items (1, 3, 5, 7, and 9) in the above array. The computed values are also stored two subscript elements apart:

```
FIRSTV:      ARRAY(11) = 0.1
DELTAV:      ARRAY(13) = 0.2
```


SECTION 9
SMOOT SUBROUTINE

GENERAL

SMOOT is a FORTRAN subroutine which draws a smooth curve through a set of data points, using a modified spline-fitting technique. The subroutine receives a pair of point coordinates from each call. Points are accumulated until a sufficient number have been received to compute a pair of cubic parametric equations for a smooth curve. This accumulation method requires the user to make an initial subroutine call and a terminal subroutine call.

CALLING SEQUENCE

CALL SMOOT (XPAGE,YPAGE,IPEN)

XPAGE,YPAGE	are the coordinates in inches (centimeters), of a single point through which the pen moves.
IPEN	determines the mode and action of the SMOOT subroutine.

Figure 9-1 illustrates the results of calls to SMOOT.

REQUIRED CALLS

The calls to SMOOT must be made in the following order:

- One initial call.
- One or more intermediate calls.
- One terminal call.

INITIAL CALL

The first call to SMOOT must use an IPEN value of 0 or -1. For this call, XPAGE, YPAGE are the X- and Y-coordinates of the first point (P_1) on the curve.

Use IPEN = 0	to initialize an open curve. The plotted curve ends at the last point (P_n) specified in a SMOOT call.
= -1	to initialize a closed curve. The plotted curve continues from the last point (P_n) back to the initial point (P_1).

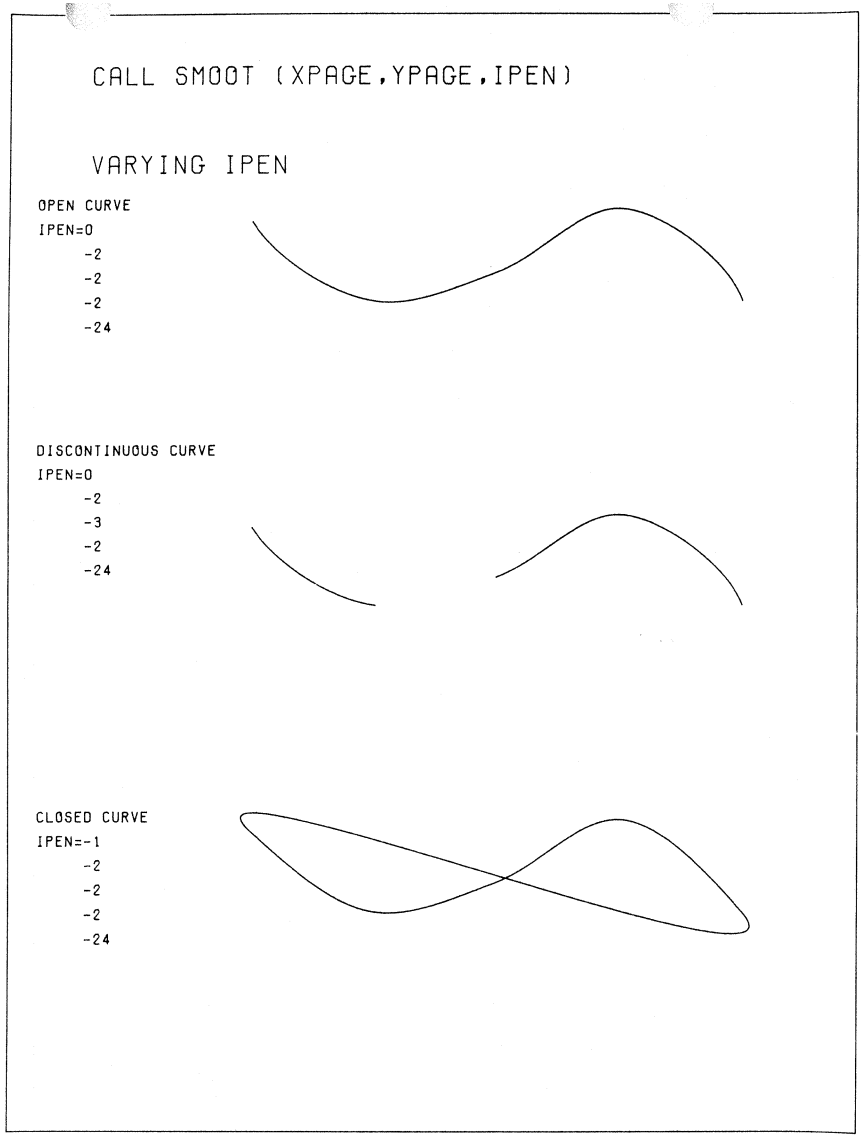


Figure 9-1. Sample of SMOOT Subroutine

INTERMEDIATE CALLS

After the initial call to SMOOT has been made, calls must be made to transfer points $P_2, P_3, \ldots P_{n-1}$, using XPAGE, YPAGE.

Use IPEN = -2	for a smoothed curve to be drawn through the points on the curve.
= -3	for the pen in the up position, to be moved through the points. The smoothing function is maintained before and after these moves.
= 2 = 3	for the call to be transferred directly to the PLOT subroutine for straight-line plotting. XPAGE, YPAGE in this call will not be considered a point on the curve. The point of departure from the SMOOT curve is the next-to-last point received by SMOOT, not the last point.

When the next call to SMOOT with IPEN = -2 or -3 is received, the pen is repositioned to the point where it left the smooth curve. The smooth curve is then continued as though the calls with IPEN = 2 or 3 had not occurred.

TERMINAL CALL

When the final point of the curve is to be communicated to SMOOT, the terminal call is made.

Use IPEN $\leq$ -24	for the terminal call. XPAGE, YPAGE are the coordinates of the final point (P_n).

After SMOOT receives a terminal call and before a subsequent initial call, any call to SMOOT with IPEN = $\pm$ 2 or $\pm$ 3 is treated as a normal call:

```
CALL PLOT(XPAGE,YPAGE,IPEN)
```

COMMENTS

Calls to other plotting subroutines may be intermixed with calls to SMOOT. Point-of-departure restrictions are the same as noted in the description above for the intermediate calls IPEN = 2 and 3.

Before a terminal call, the pen is not moved until three points on an open curve or four points on a closed curve have been received. For subsequent calls to SMOOT, the actual pen position is the next-to-last point received.

OTHER FUNCTIONAL SUBROUTINES USED

REFLX

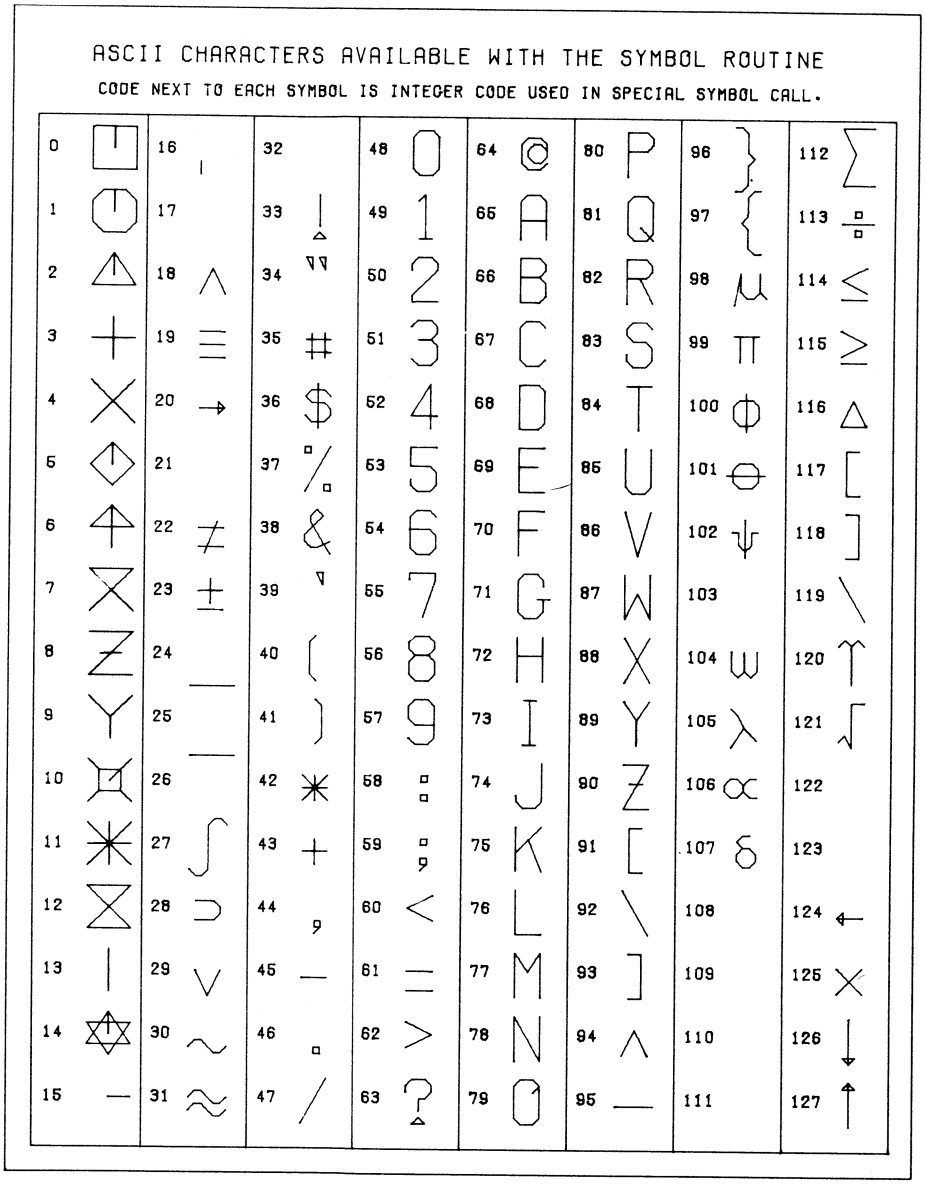


Figure 3-2. 925 Controller Standard Symbol Set

CONTENTS

ILLUSTRATIONS

CALCOMP FORGN DATA INPUT FORM

PAGE OF

1 3 NCD	NDL	5 N U	ALPHAMERIC ANNOTATION (FIELD CARD UPPER)	42 ALPHAMERIC ANNOTATION (FIELD CARD, LOWER OR BODY CONSTANT)	78 80 FCT

ALPHA CARD, NCD = -1, UP TO 72 .08" CHARS STARTING AT LEFT FORM EDGE FROM COL 6-77.
= -2, UP TO 72 .08" CHARS STARTING AFTER LAST -1 CHAR " .
= -3, UP TO 72 .14" CHARS STARTING AT LEFT FORM EDGE " .
= -4, BEFORE FORM INIT. CARD SUPPRESSES FIELD COL NRS.
= -5, BEFORE END CARD, PLOTS .5" BORDER AROUND FORM.

END CARD, NCD = 88, RESETS PLOTTER FOR NEXT FORM.
= 99, ENDS RUN.

FORM INIT CARD, NDL = NR. OF BODY LINES.

FIELD DEF CARD, NCD = NR. OF COLS IN FIELD.
NDL = NR. OF DEC+1 IN FIELD
OR IF GTR THAN FIELD+1 PLOTS
ANNO.(LWR) IN FIELD BODY.
NU. = 1, SHADES UNUSED FIELDS.

FIRST CARD ONLY, FCT. FACTORS FORM BY VALUE.

Figure 1-1. Sample FORGN Output

SECTION 1
PROGRAM CAPABILITIES

GENERAL

FORGN is a FORTRAN program which generates standard sheets with blank spaces for requested information. The forms are usable for keypunch, statistical surveys, registration, and similar applications. This user's manual describes the capabilities and gives directions for creating the desired form.

FORGN generates calls to CalComp Host Computer Basic Software (HCBS) and produces the forms on any CalComp offline or online graphics system.

TYPICAL OUTPUT FORMAT

The typical plotted form consists of three parts:

- The form title and comments, at the top of the form
- The body of the form, in the standard format determined by the input
- Comments at the bottom of the form

The first and third parts are optional. Figure 1-1 illustrates a form containing all three parts.

SIZE OF THE FORM

The normal size of the body of the plotted form is computed as follows:

Height = (Number of data lines) * 0.24 + 0.48 inches

Width = (Number of columns in a data line) * 0.13 inches

A single form containing a title line, 25 data lines, and 80 columns per line fills an area measuring 7 by 10.5 inches. No checks are made to determine whether the requested form will exceed the size of the plotting surface.

NOTE

The examples in this manual are written in inches. However, inch or centimeter units apply and are dependent on the Host Computer Basic Software (HCBS) used.

CALCOMP FORGN DATA INPUT FORM

PAGE / OF /

1 NCD	3 NDL	5 NU	ALPHAMERIC ANNOTATION (FIELD CARD UPPER)	42 ALPHAMERIC ANNOTATION (FIELD CARD, LOWER OR BODY CONSTANT)	78 80 FCT
-1					
-3			CALCOMP FORGN DATA INPUT FORM		
-1					
	23				
2			NCD		
2			NDL		
1			N	U	
36			ALPHAMERIC ANNOTATION	(FIELD CARD UPPER)	
36			ALPHAMERIC ANNOTATION	(FIELD CARD, LOWER OR BODY CONSTANT)	
3			FCT		
-1					
-1			ALPHA CARD, NCD = -1, UP TO 72 .08"	CHARS STARTING AT LEFT FORM EDGE FRO	
-2			M COL 6-77 FORM INIT CARD, NDL =	NR. OF BODY LINES	
-1			= -2, UP TO 72 .08"	CHARS STARTING AFTER LAST -1 CHAR "	
-2			FIELD DEF CARD, NCD =	NR. OF COLS IN FIELD	
-1			= -3, UP TO 72 .14"	CHARS STARTING AT LEFT FORM EDGE "	
-2			NDL =	NR. OF DEC+1 IN FIELD	
-1			= -4, BEFORE FORM INIT	CARD SUPPRESSES FIELD COL NRS	
-2				OR IF GTR THAN FIELD+1 PLOTS	
-1			= -5, BEFORE END CARD, PLOTS .5"	BORDER AROUND FORM	
-2				ANNO.(LWR) IN FIELD BODY	
-1			END CARD, NCD = 88, RESETS PLOTTER	FOR NEXT FORM	
99					

ALPHA CARD, NCD - -1, UP TO 72 .08" CHARS STARTING AT LEFT FORM EDGE FROM COL 6-77.
- -2, UP TO 72 .08" CHARS STARTING AFTER LAST -1 CHAR " .
- -3, UP TO 72 .14" CHARS STARTING AT LEFT FORM EDGE " .
- -4, BEFORE FORM INIT. CARD SUPPRESSES FIELD COL NRS.
- -5, BEFORE END CARD, PLOTS .5" BORDER AROUND FORM.

END CARD, NCD - 88, RESETS PLOTTER FOR NEXT FORM.
- 99, ENDS RUN.

FORM INIT CARD, NDL - NR. OF BODY LINES.
FIELD DEF CARD, NCD - NR. OF COLS IN FIELD.
NDL - NR. OF DEC+1 IN FIELD
OR IF GTR THAN FIELD+1 PLOTS
ANNO.(LWR) IN FIELD BODY.
NU. - 1, SHADES UNUSED FIELDS.
FIRST CARD ONLY, FCT. FACTORS FORM BY VALUE.

Figure 2-1. Sample FORGN Input

SECTION 2
INPUT FORMAT

GENERAL

Figure 2-1 illustrates a typical input sequence. This sequence was used to create Figure 1-1.

CARD TYPES

FORGN uses four types of Data Cards: Alpha Cards, Form Initiator Cards, Field Definition Cards, and End Cards.

1. Alpha Card – prints alphameric information at the top or bottom of a form; suppresses plotting of column numbers; or causes a border to be plotted around the form.

   Alpha Cards may be used only before a Form Initiator Card and after the last of a group of Field Definition Cards. The information on the first Alpha Card of a run and on the first Alpha Card following an End Card causes plotting to start at an unfactored Y coordinate of 10.3 inches, and at the X coordinate corresponding to the pen's current position.

2. Form Initiator Card – defines the number of data lines in the body of the form.

3. Field Definition Card – defines the width of a field and the annotation describing the field. There must be one Field Definition Card for each field in the form.

4. End Card – signals the end-of-page condition and immediately precedes additional form generation or termination of the run.

CONTROLLING THE SIZE OF THE FORM

The form can be scaled up or scaled down by entering a scale factor (FORTRAN Format F3.2) in columns 78-80 of the first card of the deck used for generating the form. If these columns are left blank or contain zeros, a scale factor of 1.0 will be assumed.

ALPHA CARD

The purpose of the Alpha Card is to specify the plotting of alphameric information at the top and/or bottom of the form. The following is a list of the data by card columns:

ALPHA CARD (cont)

Card Columns	FORTRAN Format	Field Name	Description
1-2	I2	NCD	Plots 72 characters from the BCD field as follows: If NCD = -1, the characters are 0.08 inches high and start at the left edge of the form, 0.02 inches below the preceding line, if any. If NCD = -2, the characters are 0.08 inches high and start at 5.76 inches from the left edge of the form, i.e., where a previous NCD = -1 alpha field would leave off. 144 characters 0.08 inches high can be plotted on one line. If NCD = -3, the characters are 0.16 inches high and start at the left edge of the form. If NCD = -4, column numbers normally plotted at the beginning of a field are suppressed, and the BCD field is ignored. An Alpha Card punched this way must come before the Form Initiator Card. If NCD = -5, a 0.5-inch border is drawn around the completed form. An Alpha Card punched this way must be the last data card of a form, and must come just before the End Card.
3-5			(Blank)
6-77	A	BCD	Alphameric information to be plotted according to the NCD Code. This field has no effect if NCD = -4 or -5.
78-80			(Blank)

FORM INITIATOR CARD

The Form Initiator Card supplies the number of lines on the form. There must be one Form Initiator Card for each form. Every Form Initiator Card must be followed by at least one Field Definition Card.

Card Columns	FORTRAN Format	Field Name	Description
1-2			(Blank)
3-4	I2	N	Integer number of data transmittal lines in the body of the form.
5-80			(Blank)

MULTIPLE FORMS ON A SINGLE PAGE

Mulitple forms can be "stacked" on a single page by positioning a new Form Initiator Card immediately following the data cards for the previous form. The new form is plotted immediately below the previous form. No check is made to see if the new form will exceed the physical page size.

Figure 2-2 shows multiple-form segments on a page. Figures 2-3 and 2-4 show the input required to produce Figure 2-2.

FIELD DEFINITION CARD

Card Columns	FORTRAN Format	Field Name	Description
1-2	I2	NCOL	Integer number of columns in the field (maximum of 80).
3-4	I2	NDEL	One plus the integer number of decimal places in this field. (This field causes a caret to be plotted to indicate an implied decimal point.) If NDEL is less than or equal to NCOL, BCDL is a second line of alphameric field-description data to be plotted below the first line. If NDEL is two plus the number of columns (NCOL), the characters in the BCDL field are plotted on each data line in the body of the form.
5	I1	NU	Shading flag. If NU is blank or zero, no shading is plotted. If NU = 1, the field is shaded to indicate that no data is to be entered in that field.
6-41	A	BCDU	Alphameric field-description data to be plotted at the top of the body of the form, over the field.
42-77	A	BCDL	If NDEL is less than or equal to NCOL, this field is a second line of alphameric field-description data to be plotted below the first line. If NDEL is two plus NCOL, this field is a constant string to be plotted on each data line.
78-80			(Blank)

The maximum number of characters, NC, in BCDU or BCDL is described in the following expression:

NC = 2 * NCOL - 1

FØRGN EXAMPLE ØF MULTIPLE FØRM SEGMENTS ØN ØNE FØRM

KEYPUNCH - GANG THIS INFØ INTØ EACH CARD

1 IDENT CØDE	8 MØ	10 DA	12 YR

CARD TYPE 1

1 C		14 FIELD A DESCRIPTIØN - LINE ØNE LINE TWØ	34 FIELD B DESCRIPTIØN - LINE ØNE LINE TWØ	76 JØB IDENT 80
1				
1				
1				
1				
1				
1				

CARD TYPE 2

1 C		14 A	20 B	26 C	32 D	38 E	44 F	50 G	56 H	62 I	68 FIELD CONSTANT	76 JØB IDENT 80
2											CALCØMP	
2											CALCØMP	
2											CALCØMP	
2											CALCØMP	
2											CALCØMP	
2											CALCØMP	
2											CALCØMP	
2											CALCØMP	
2											CALCØMP	
2											CALCØMP	
2											CALCØMP	
2											CALCØMP	

FØRM 13-80 REVISED 01-30-80

Figure 2-2. Sample Multiple Form Output

END CARD

Card Columns	FORTRAN Format	Field Name	Description
1-2	I2	NCD	End-of-Page indicator. If NCD = 88, FORGN reorigins the plot to a new page and reads an input data line for the next form. If NCD = 99, FORGN reorigins the plot to a new page and terminates the run.
3-80			(Blank)

CARD SEQUENCE ERRORS

At least one Field Description Card must follow every Form Initiator Card. If the Field Description Cards are missing, an error message, "Card Sequence Error Detected", will be written on the computer system's print tape or console typewriter.

There are no other restrictions on the quantity or order of data cards.

SUBROUTINES INCLUDED IN FORGN

MAIN
SYMBOL
SYMDE
SCURV
FLEX
NUMBER

CALCOMP FORGN DATA INPUT FORM

PAGE 1 OF 2

1 3 NCD	NDL	5 N U	ALPHAMERIC ANNOTATION (FIELD CARD UPPER)	42 ALPHAMERIC ANNOTATION (FIELD CARD, LOWER OR BODY CONSTANT)	78 80 FCT
-1					
-3			FORGN EXAMPLE OF MULTIPLE FORM SEGMENTS ON ONE FORM		
-1					
-1			KEYPUNCH - GANG THIS INFO INTO EACH	CARD	
-1					
	1				
1		1			
6			IDENT	CODE	
2			MO		
2			DA		
2			YR		
-3					
-1			CARD TYPE 1		
-1					
	6				
1	3		C	1	
12		1			
20			FIELD A DESCRIPTION - LINE ONE	LINE TWO	
42			FIELD B DESCRIPTION - LINE ONE	LINE TWO	
5			JOB	IDENT	
-3					
-1			CARD TYPE 2		
-1					

ALPHA CARD, NCD - -1, UP TO 72 .08" CHARS STARTING AT LEFT FORM EDGE FROM COL 6-77.
- -2, UP TO 72 .08" CHARS STARTING AFTER LAST -1 CHAR " .
- -3, UP TO 72 .14" CHARS STARTING AT LEFT FORM EDGE " .
- -4, BEFORE FORM INIT. CARD SUPPRESSES FIELD COL NRS.
- -5, BEFORE END CARD, PLOTS .5" BORDER AROUND FORM.

END CARD, NCD - 88, RESETS PLOTTER FOR NEXT FORM.
- 99, ENDS RUN.

FORM INIT CARD, NDL - NR. OF BODY LINES.
FIELD DEF CARD, NCD - NR. OF COLS IN FIELD.
NDL - NR. OF DEC+1 IN FIELD
OR IF GTR THAN FIELD+1 PLOTS
ANNO.(LWR) IN FIELD BODY.
NU. - 1, SHADES UNUSED FIELDS.
FIRST CARD ONLY, FCT. FACTORS FORM BY VALUE.

Figure 2-3. Sample Multiple Form Input

CALCOMP FORGN DATA INPUT FORM

PAGE 2 OF 2

NCD	NDL	NU	ALPHAMERIC ANNOTATION (FIELD CARD UPPER)	ALPHAMERIC ANNOTATION (FIELD CARD, LOWER OR BODY CONSTANT)	FCT
	12				
1	3		C	2	
12		1			
6			A		
6			B		
6			C		
6			D		
6			E		
6			F		
6			G		
6			H		
6			I		
81	10		FIELD CONSTANT	CALCOMP	
5			JOB	IDENT	
-1					
-1			FORM 13-80 REVISED 01-31-80		
-5					
99					

ALPHA CARD, NCD = -1, UP TO 72 .08" CHARS STARTING AT LEFT FORM EDGE FROM COL 6-77.
= -2, UP TO 72 .08" CHARS STARTING AFTER LAST -1 CHAR " .
= -3, UP TO 72 .14" CHARS STARTING AT LEFT FORM EDGE " .
= -4, BEFORE FORM INIT. CARD SUPPRESSES FIELD COL NRS.
= -5, BEFORE END CARD, PLOTS .5" BORDER AROUND FORM.

END CARD, NCD = 88, RESETS PLOTTER FOR NEXT FORM.
= 99, ENDS RUN.

FORM INIT CARD, NDL = NR. OF BODY LINES.

FIELD DEF CARD, NCD = NR. OF COLS IN FIELD.
NDL = NR. OF DEC+1 IN FIELD OR IF GTR THAN FIELD+1 PLOTS ANNO.(LWR) IN FIELD BODY.
NU. = 1, SHADES UNUSED FIELDS.

FIRST CARD ONLY, FCT. FACTORS FORM BY VALUE.

Figure 2-4. Sample Multiple Form Input

CONTENTS

ILLUSTRATIONS

SECTION 1
INTRODUCTION

GENERAL

This user's manual describes the calling sequence and arguments for the General Applications category of CalComp's Functional Software Library. This category is comprised of eight standard FORTRAN subroutines, which generate calls to CalComp Host Computer Basic Software (HCBS).

Each subroutine description includes plotted-output samples which show the effects of using various argument values. The argument names used in the calling sequences are arbitrary and need not be used. They have been chosen to illustrate the use of the argument and its particular type, i.e., if the initial of the name is I - N, the argument is an Integer type; otherwise, it is a Real type.

Any other Functional subroutines called by the subroutines described herein are supplied with the category package, but are not intended to be called independently.

FUNCTIONAL SUBROUTINES

The subroutine descriptions are arranged alphabetically, following a composite sample plot (see Figure 1) that illustrates the use of all eight subroutines:

CIRCL	–	draws a circle, arc, or spiral
DASHL	–	draws dashed lines connecting a series of data points
DASHP	–	draws a dashed line to a specified point
ELIPS	–	draws an ellipse or elliptical arc
FIT	–	draws a curve through three points
GRID	–	draws a linear grid
POLY	–	draws an equilateral polygon
RECT	–	draws a rectangle.

NOTE

This manual applies to both inch and metric measurement. Metric parameter values are shown in parenthesis in all illustrations. Metric or inch measurement is dependent upon the specific Host Computer Basic Software being used.

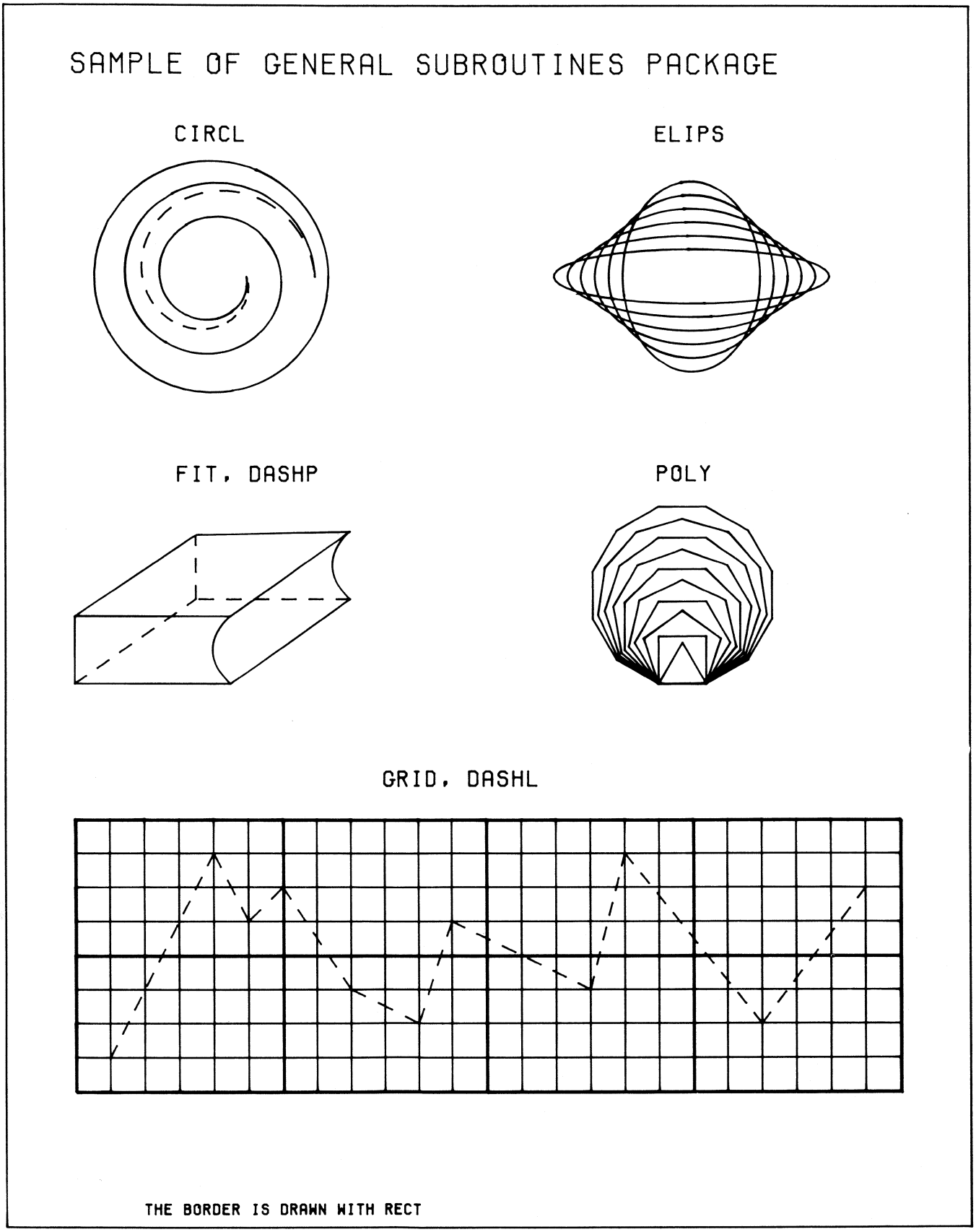


Figure 1. Sample of GENERAL Subroutines Usage

Please Use "CIRCLE"
zie supplement pag. 2

SECTION 2
CIRCL SUBROUTINE

GENERAL

CIRCL is a FORTRAN subroutine that draws an arc which may be extended to form a circle or spiral. This arc is started at the given point XPAGE, YPAGE.

CALLING SEQUENCE

CALL CIRCL (XPAGE, YPAGE, THO, THF, RO, RF, DI)

XPAGE, YPAGE	are the coordinates, in inches (centimeters), of the arc's starting point.
THO	is the radius angle, in degrees (counterclockwise), from the X-axis, for the start of the arc.
THF	is the radius angle, in degrees (counterclockwise), from the X-axis, for the end of the arc.
RO	is the arc's starting radius, in inches (centimeters) measured from XPAGE, YPAGE.
RF	is the arc's ending radius, in inches (centimeters) measured from XPAGE, YPAGE.
DI	is a code used to specify the type of line desired.
	If DI = 0.0, a solid arc is drawn. If DI = 0.5, a dashed arc is drawn.

COMMENTS

THO and THF may be positive or negative. If THO is less than THF, the arc is drawn in a counterclockwise direction. If THO is greater than THF, the arc is drawn in a clockwise direction.

OTHER FUNCTIONAL SUBROUTINES USED

None.

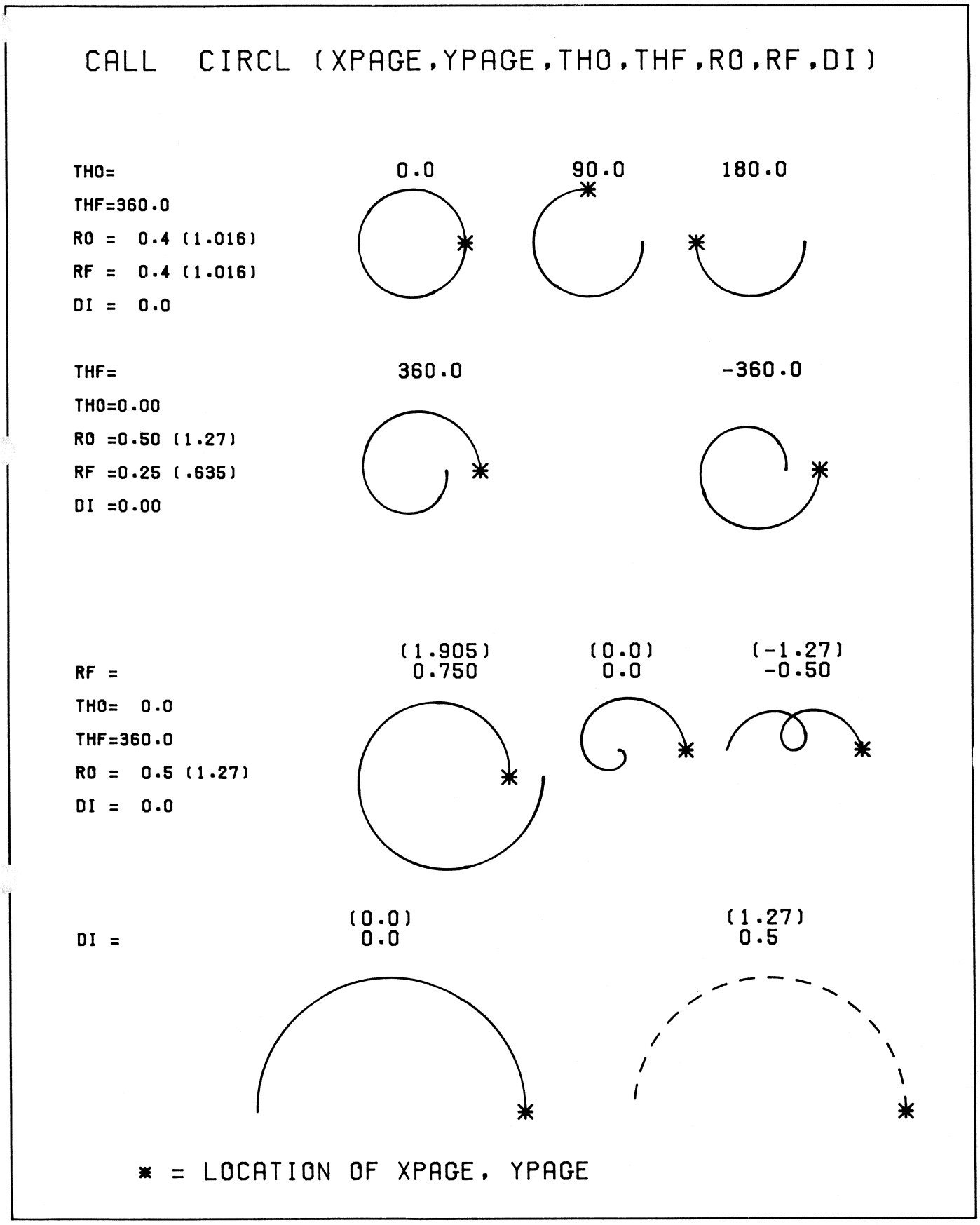


Figure 2. Sample of CIRCL Subroutines Usage

SECTION 3
DASHL SUBROUTINE

GENERAL

DASHL is a FORTRAN subroutine which draws dashed lines connecting a series of data points.

CALLING SEQUENCE

CALL DASHL (XARRAY,YARRAY,NPTS,INC)

XARRAY	is the name of the array containing the abscissas of the data points to be plotted.
YARRAY	is the name of the array containing the ordinates of the data points to be plotted.
NPTS	is the quantity of data points to be plotted.
INC	is the increment between array elements. INC is greater than 1 if the values to be plotted are in a mixed or multidimensioned array. (Normally, INC=1.)

COMMENTS

The arrays must be dimensioned so as to contain the adjusted minimum value (FIRSTV) and the adjusted delta value (DELTAV). These may be provided by the Host Computer Basic Software (HCBS) SCALE subroutine and are stored in the data array following the data.

For the X-array, the adjusted minimum is stored in XARRAY (NPTS*INC+1), and the adjusted delta is in XARRAY (NPTS*INC+INC+1). Similarly, for the Y-array, the minimum is in YARRAY (NPTS*INC+1),and the delta is in YARRAY (NPTS*INC+INC+1).

If the SCALE subroutine is not used, the user must place the appropriate adjusted minimum values and the adjusted delta values in the specified elements of the arrays. For a one-to-one correspondence between array data and plotted data, these values should be 0.0 (minimum) and 1.0 (delta).

A dashed line with dashes approximately 0.1 inch (0.254 centimeter) long, is drawn connecting sequential points. Coding is optimized so that plotting may either begin at the first point and progress forward or begin at the last point and progress backward.

OTHER FUNCTIONAL SUBROUTINES USED

None.

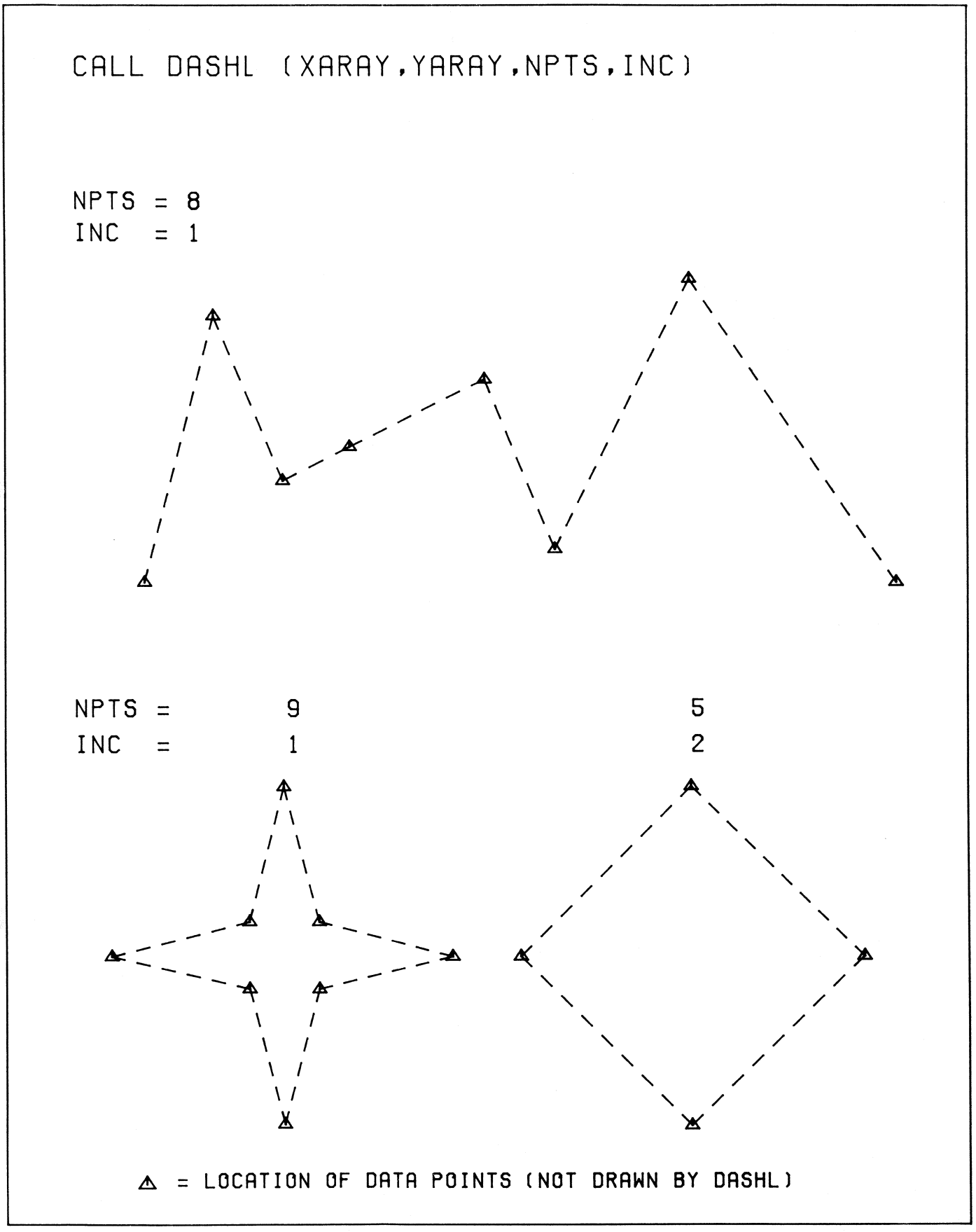


Figure 3. Sample of DASHL Subroutines Usage

SECTION 4
DASHP SUBROUTINE

GENERAL

DASHP is a FORTRAN subroutine which draws a dashed line from the pen's present position to the specified point (XPAGE, YPAGE).

CALLING SEQUENCE

CALL DASHP (XPAGE,YPAGE,DASH)

XPAGE,YPAGE	are the coordinates, in inches (centimeters), of the point to which the dashed line is to be drawn.
DASH	is the length, in inches (centimeters), of each dash and of the space between dashes.

COMMENTS

If the line length is less than double the dash length, the dash length is adjusted to half the line length.

OTHER FUNCTIONAL SUBROUTINES USED

None.

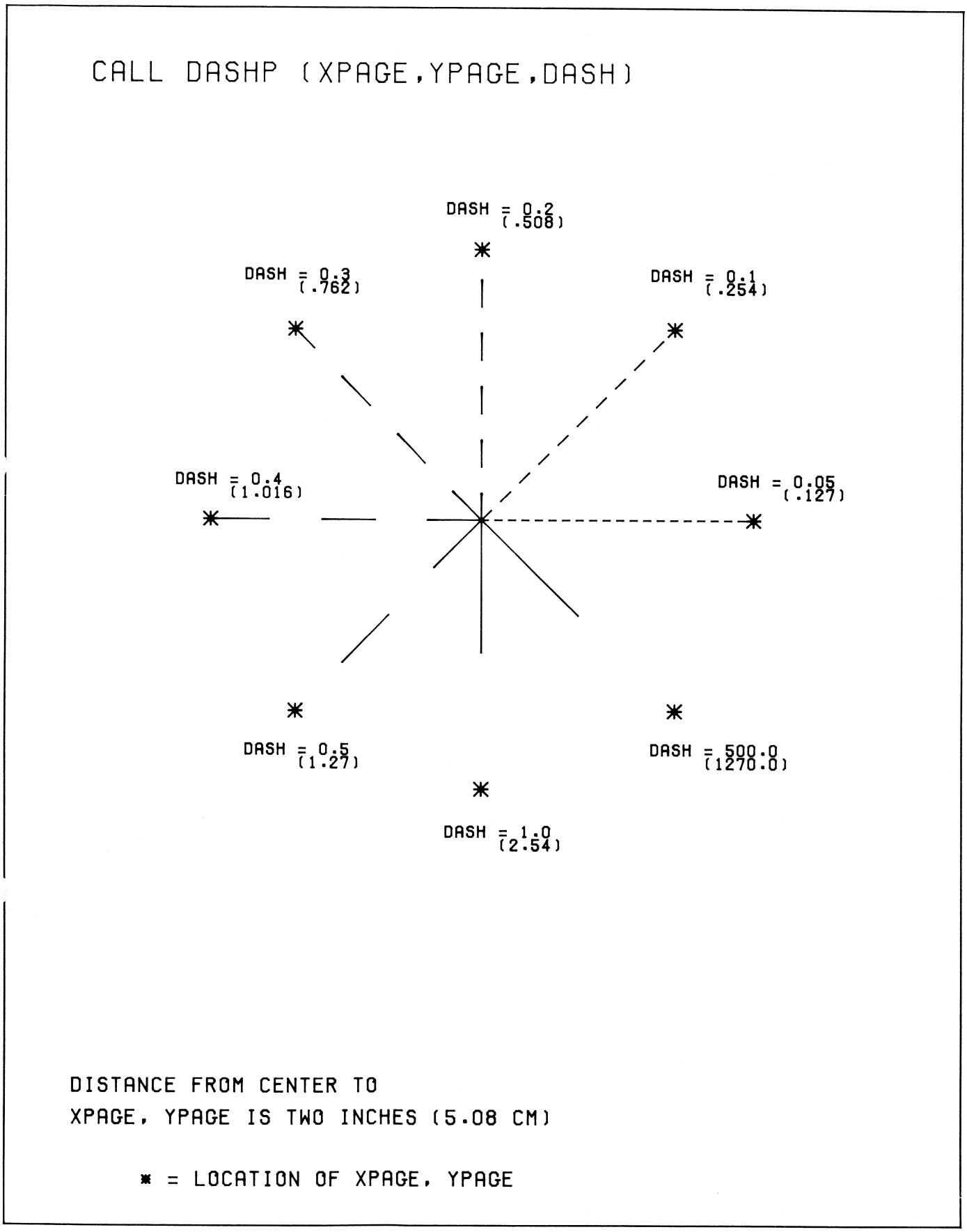


Figure 4. Sample of DASHP Subroutines Usage

SECTION 5
ELIPS SUBROUTINE

GENERAL

ELIPS is a FORTRAN subroutine which draws an ellipse or elliptical arc.

CALLING SEQUENCE

CALL ELIPS (XPAGE,YPAGE,RMAJ,RMIN,ANGLE,THO,THF,IPEN)

XPAGE,YPAGE	are the coordinates, in inches (centimeters), of the starting point of the ellipse or arc.
RMAJ,RMIN	are the lengths, in inches (centimeters), of the semimajor and semiminor axes, respectively.
ANGLE	is the angle of the major axis, in degrees (counterclockwise) from the X-axis.
THO,THF	are the angles, in degrees (counterclockwise) from the X-axis with respect to ANGLE, of the arc's starting and ending points.
IPEN	is the pen code used while moving the pen to the arc's starting point. If the value of IPEN is 3, the pen is up for the move. If the value of IPEN is 2, the pen is down for the move.

COMMENTS

THO and THF may be positive or negative. If THO is less than THF, the arc is drawn in a counterclockwise direction. If THO is greater than THF, the arc is drawn in a clockwise direction.

OTHER FUNCTIONAL SUBROUTINES USED

None.

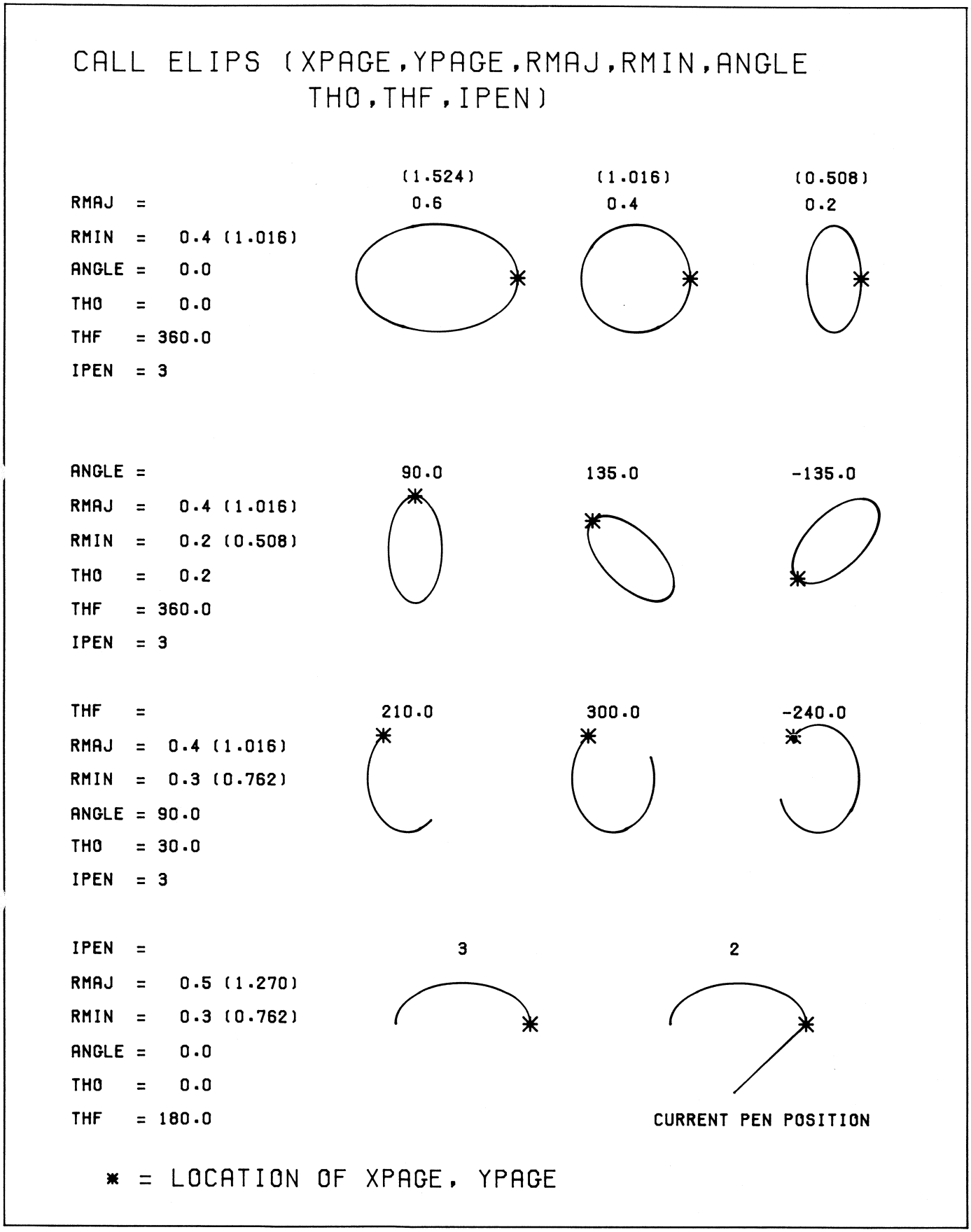


Figure 5. Sample of ELIPS Subroutines Usage

SECTION 6
FIT SUBROUTINE

GENERAL

FIT is a FORTRAN subroutine which draws a semihyperbolic curve through three points.

CALLING SEQUENCE

CALL FIT (XPAGE1,YPAGE1,XPAGE2,YPAGE2,XPAGE3,YPAGE3)

XPAGE1,YPAGE1 are the X and Y coordinates, in inches (centimeters), of the
XPAGE2,YPAGE2 three points through which the curve passes.
XPAGE3,YPAGE3

COMMENTS

This subroutine generates a semihyperbolic fit using the three given points. A set of points for which a fit is not possible is drawn with straight-line segments.

RESTRICTIONS

The curve through the three points must not be multivalued in both X and Y. The middle point (XPAGE2,YPAGE2) must be between the end points along the X-axis or the Y-axis.

XPAGE1<XPAGE2<XPAGE3

or

XPAGE1>XPAGE2>XPAGE3

or

YPAGE1<YPAGE2<YPAGE3

or

YPAGE1>YPAGE2>YPAGE3

OTHER FUNCTIONAL SUBROUTINES USED

SOLUT.

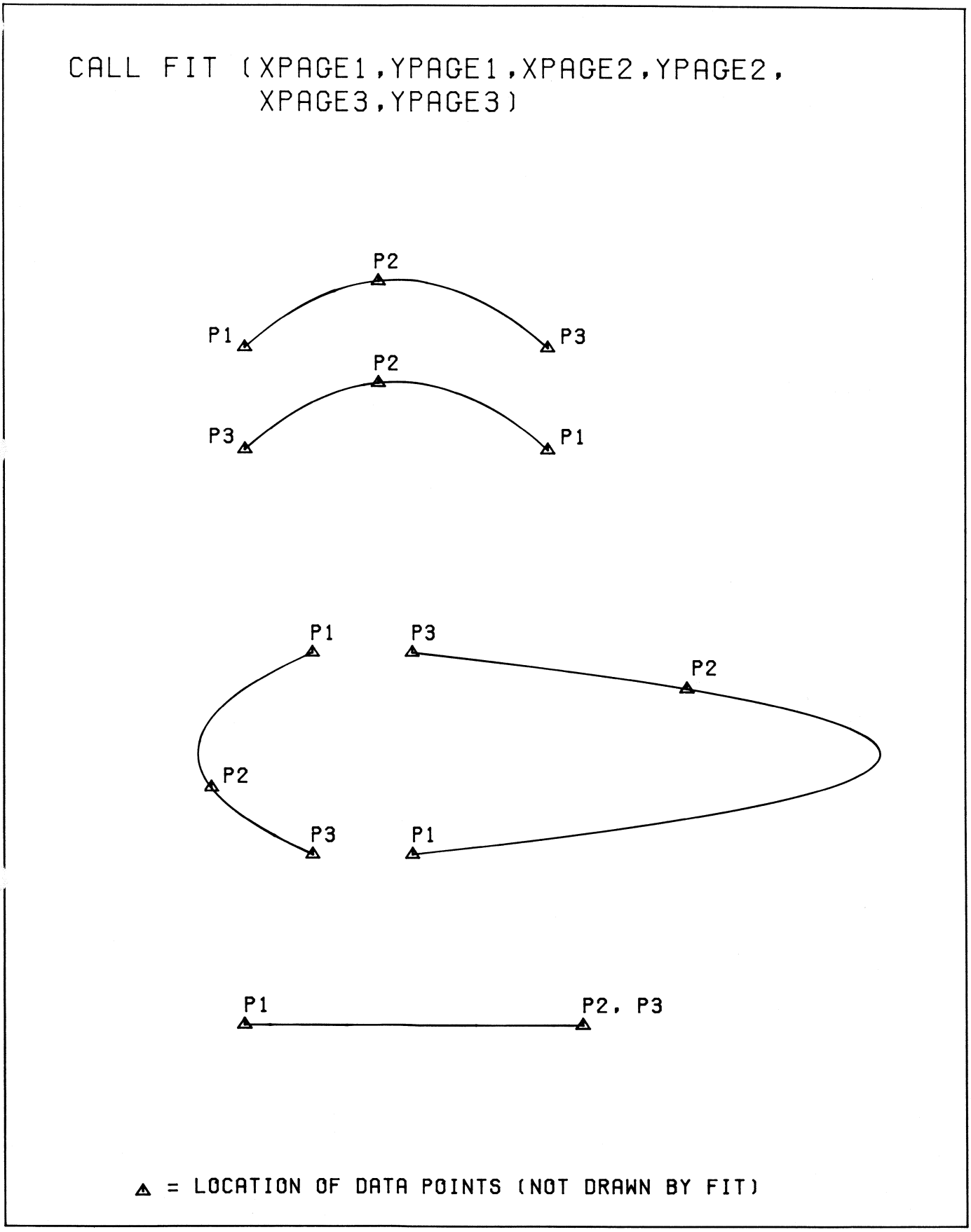


Figure 6. Sample of FIT Subroutines Usage

SECTION 7
GRID SUBROUTINE

GENERAL

GRID is a FORTRAN subroutine which draws a linear grid.

CALLING SEQUENCE

CALL GRID (XPAGE,YPAGE,DELTAX,DELTAY,NXSP,NYSP)

XPAGE,YPAGE	are the coordinates, in inches (centimeters), of the grid's lower left corner.
DELTAX	is the number of inches (centimeters), between grid lines in the X direction.
DELTAY	is the number of inches (centimeters), between grid lines in the Y direction.
NXSP,NYSP	are the number of spaces between lines in the X and Y directions, respectively.

COMMENTS

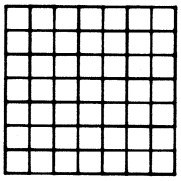
GRID generates a linear grid of any size. The number of lines drawn is NXSP + 1 in the X direction and NYSP + 1 in the Y direction.

OTHER FUNCTIONAL SUBROUTINES USED

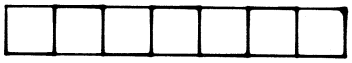
None.

CALL GRID (XPAGE,YPAGE,DELTAX,DELTAY,NXSP,NYSP)

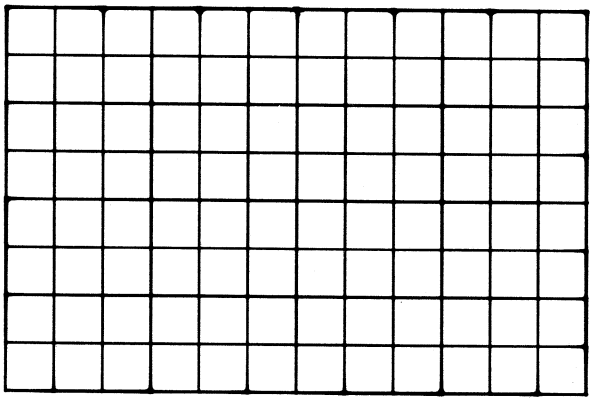
DELTAX = 0.125 (0.3175)
DELTAY = 0.125 (0.3175)
NXSP = 7
NYSP = 7



DELTAX = 0.25 (0.635)
DELTAY = 0.25 (0.635)
NXSP = 7
NYSP = 1



DELTAX = 0.25 (0.635)
DELTAY = 0.25 (0.635)
NXSP = 12
NYSP = 8
LINES ARE ACCENTED BY
OVERLAYING THE FIRST
GRID WITH ANOTHER
(SECOND GRID)
DELTAX = 0.75 (1.905)
DELTAY = 0.50 (1.270)
NXSP = 4
NYSP = 4



XPAGE AND YPAGE ARE OFFSET 0.01 INCHES (.0254 CM)

Figure 7. Sample of GRID Subroutines Usage

SECTION 8
POLY SUBROUTINE

GENERAL

POLY is a FORTRAN subroutine used to draw equilateral polygons.

CALLING SEQUENCE

CALL POLY (XPAGE,YPAGE,SLEN,SN,ANGLE)

XPAGE,YPAGE	are the coordinates, in inches (centimeters), of the polygon's starting point.
SLEN	is the length, in inches (centimeters), of a side of the polygon.
SN	is the number of sides of the polygon (fractional values are truncated to integers).
ANGLE	is the angle, in degrees (counterclockwise), from the X-axis of the first side of the polygon.

COMMENTS

If SN is negative, a star is drawn with SN points.

OTHER FUNCTIONAL SUBROUTINES USED

None.

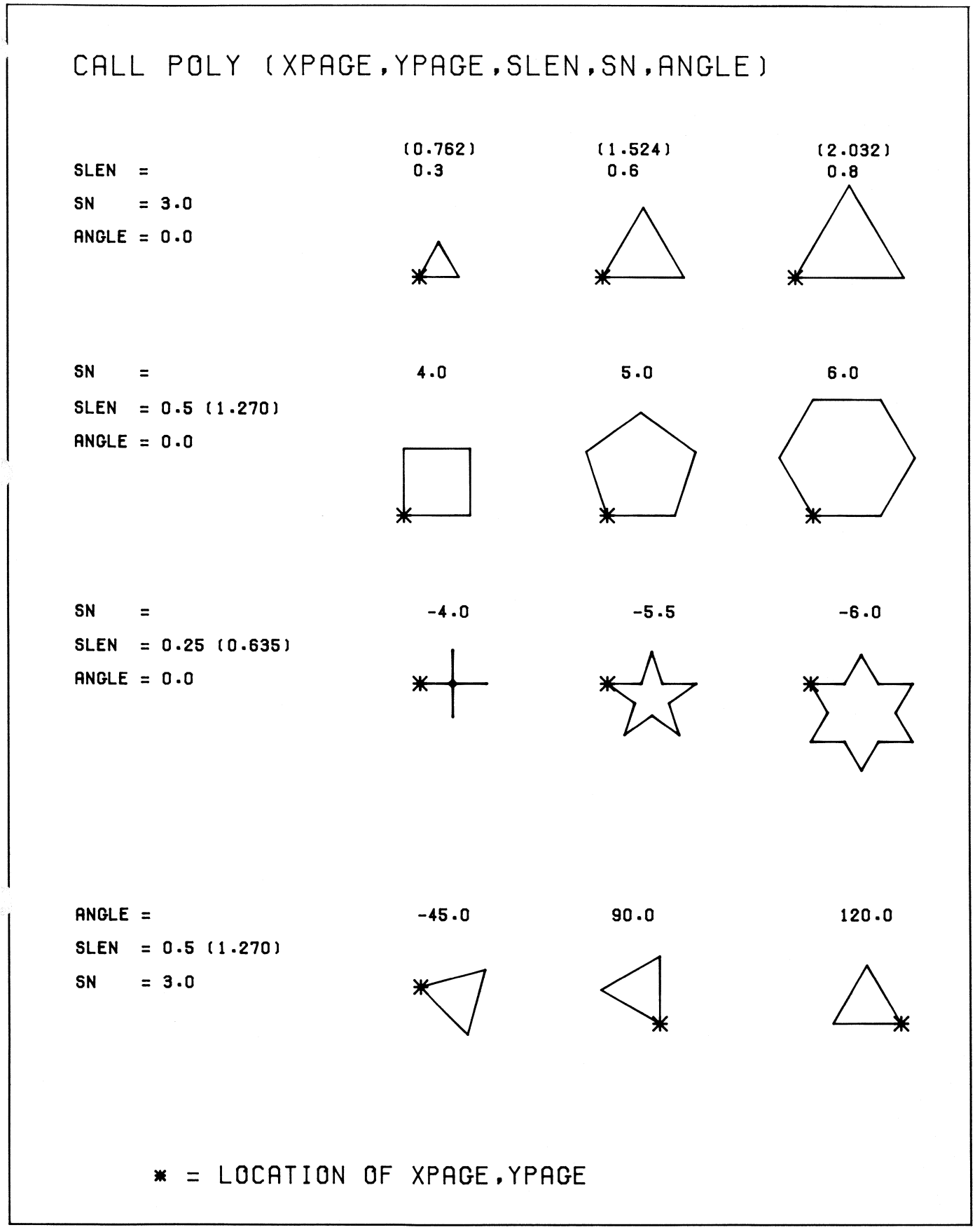


Figure 8. Sample of POLY Subroutines Usage

SECTION 9
RECT SUBROUTINE

GENERAL

RECT is a FORTRAN subroutine used to draw rectangles.

CALLING SEQUENCE

CALL RECT (XPAGE,YPAGE,HEIGHT,WIDTH,ANGLE, IPEN)

XPAGE,YPAGE	are the coordinates, in inches (centimeters), of the rectangle's lower left corner, before rotation.
HEIGHT	is the rectangle's height, in inches (centimeters).
WIDTH	is the rectangle's width, in inches (centimeters).
ANGLE	is the angle, in degrees (counterclockwise), from the X-axis at which the rectangle's base is to be drawn. (Rectangle is rotated about XPAGE,YPAGE.)
IPEN	is the code used while moving the pen to the rectangle's starting point.
	If IPEN = 3, the pen is up for the move.
	If IPEN = 2, the pen is down for the move.

OTHER FUNCTIONAL SUBROUTINES USED

None.

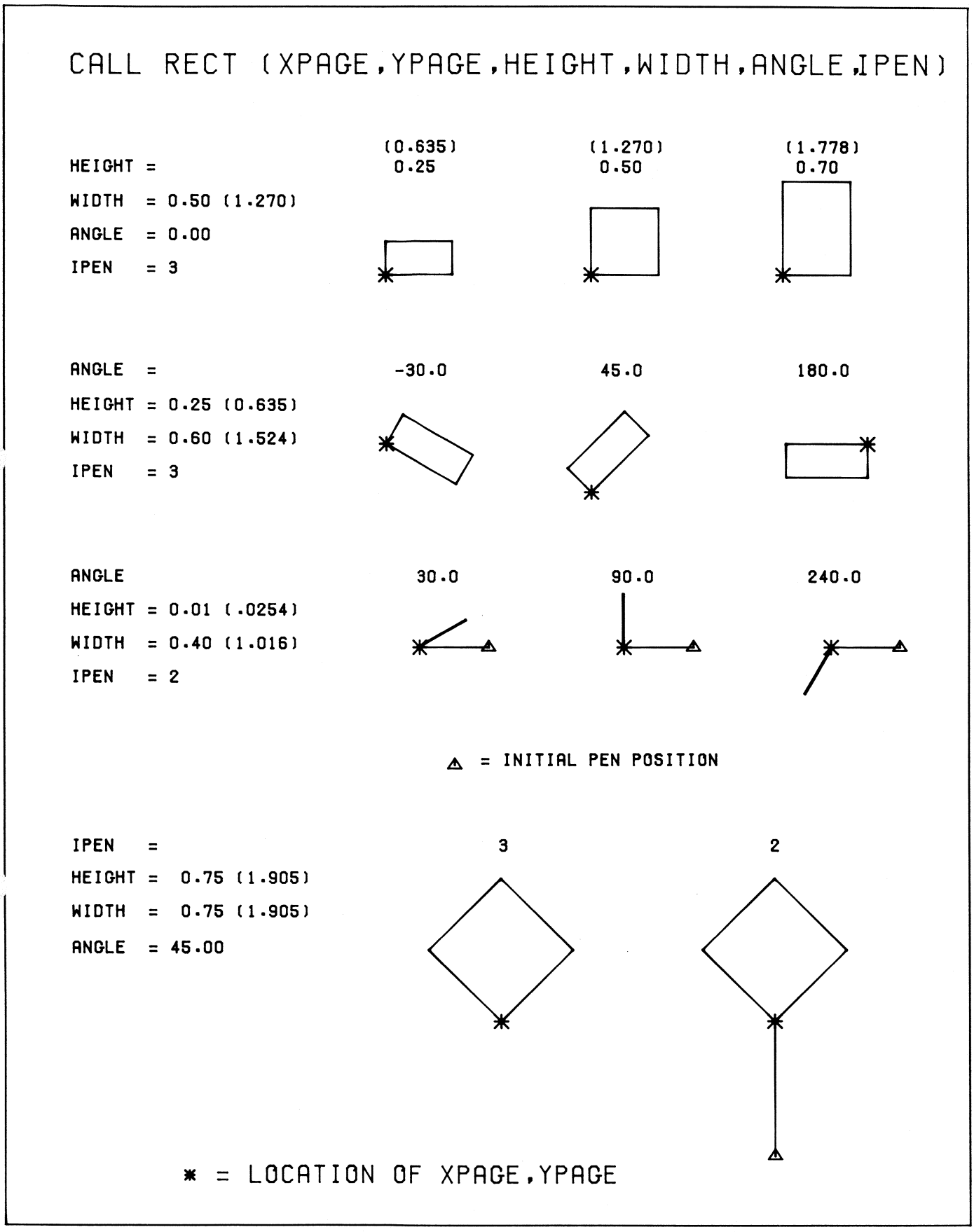


Figure 9. Sample of RECT Subroutines Usage

CHARACTERS AVAILABLE IN SYMBOL ROUTINE FOR CALCOMP 907 AND IBM 370 COMPUTER

0		16		32	0	48	@	64	P	80	°
1		17	!	33	1	49	A	65	Q	81	a
2		18	"	34	2	50	B	66	R	82	b
3		19	#	35	3	51	C	67	S	83	c
4		20	$	36	4	52	D	68	T	84	d
5		21	%	37	5	53	E	69	U	85	e
6		22	&	38	6	54	F	70	V	86	f
7		23	'	39	7	55	G	71	W	87	g
8		24	(	40	8	56	H	72	X	88	h
9		25	)	41	9	57	I	73	Y	89	i
10		26	*	42	:	58	J	74	Z	90	j
11		27	+	43	;	59	K	75	[	91	k
12		28	,	44	<	60	L	76	\	92	l
13		29	—	45	=	61	M	77	]	93	m
14		30	.	46	>	62	N	78	↑	94	n
15	—	31	/	47	?	63	O	79	←	95	o

INTEGER FOR USE IN SYMBOL CALL SHOWN TO LEFT OF EACH SYMBOL

CHARACTERS AVAILABLE IN SYMBOL ROUTINE FOR CALCOMP 907 AND IBM 370 COMPUTER

96	p	112		128	8	144	−	160	π	176	≠
97	q	113	¢	129	9	145	α	161	ρ	177	±
98	r	114	¤	130	0	146	β	162	σ	178	∫
99	s	115	®	131	+	147	γ	163	τ	179	⊃
100	t	116	▩	132	−	148	δ	164	υ	180	△
101	u	117	§	133	1	149	ζ	165	φ	181	—
102	v	118	©	134	2	150	η	166	χ	182	⊂
103	w	119	▷	135	3	151	ϵ	167	ψ	183	√
104	x	120	⫸	136	4	152	θ	168	ω	184	∞
105	y	121	1	137	5	153	ι	169	∨	185	⌐
106	z	122	2	138	6	154	κ	170	∧	186	∝
107	{	123	3	139	7	155	λ	171	≡	187	∴
108	|	124	4	140	8	156	μ	172	→	188	≤
109	}	125	5	141	9	157	ν	173	≈	189	≥
110	~	126	6	142	0	158	ξ	174	×	190	⋮
111	`	127	7	143	+	159	ο	175	÷	191	∘

INTEGER FOR USE IN SYMBOL CALL SHOWN TO LEFT OF EACH SYMBOL

CHARACTERS AVAILABLE IN SYMBOL ROUTINE FOR CALCOMP 907 AND IBM 370 COMPUTER

192	⊤	208		224		240	Ö	256	ä	272	
193	~	209	≠	225		241	Ø	257	ö	273	
194	⌇	210	-	226		242	Ü	258	ü	274	
195	▫	211	?	227		243		259		275	
196	△	212	▫	228		244	Ñ	260		276	
197	▽	213	=	229		245		261		277	
198	▯	214		230		246		262		278	△
199	⊥	215		231		247		263		279	
200	⌊	216		232		248	ß	264	£	280	
201	⌈	217	—	233		249		265		281	
202	○	218	=	234		250	æ	266		282	
203	|	219		235		251	ø	267		283	
204	✶	220		236		252		268		284	
205		221		237	Ä	253	ç	269		285	
206	∪	222		238	Å	254	œ	270		286	
207	∩	223		239	Æ	255		271	|	287	Ω

INTEGER FOR USE IN SYMBOL CALL SHOWN TO LEFT OF EACH SYMBOL

CONTENTS

ILLUSTRATIONS

SECTION 1
INTRODUCTION

GENERAL

This user's manual describes the calling sequences and arguments for the Business Applications category of CalComp's Functional Software Library. These seven standard FORTRAN subroutines generate calls to CalComp Host Computer Basic Software (HCBS).

Each subroutine description includes plotted-output samples which show the effects of using various argument values. The argument names used in the calling sequences are arbitrary and need not be used. They have been chosen to illustrate the use of the argument and its particular type, i.e., if the initial of the name is I - N, the argument is an Integer type; otherwise, it is a Real type.

Any other Functional subroutines called by the subroutines described herein are supplied with the category package, but are not intended to be called independently.

FUNCTIONAL SUBROUTINES

The subroutine descriptions are arranged alphabetically, using sample plots. These plots illustrate the use of the subroutines, e.g., see Figures 1-1 and 1-2.

AXISB — draws an axis with business-oriented annotation

AXISC — draws an axis with calendar-month annotation

BAR — draws bars for bar-graph plotting

LBAXS — draws a logarithmic axis with business annotation

LGLIN — plots data either in log-log or in semi-log mode

SCALG — performs scaling for logarithmic plotting

SHADE — draws shading between designated lines

NOTE

This manual applies to both inch and metric measurement. Metric parameter values are shown in parentheses in all illustrations. Metric or inch measurement is dependent upon the specific Host Computer Basic Software being used.

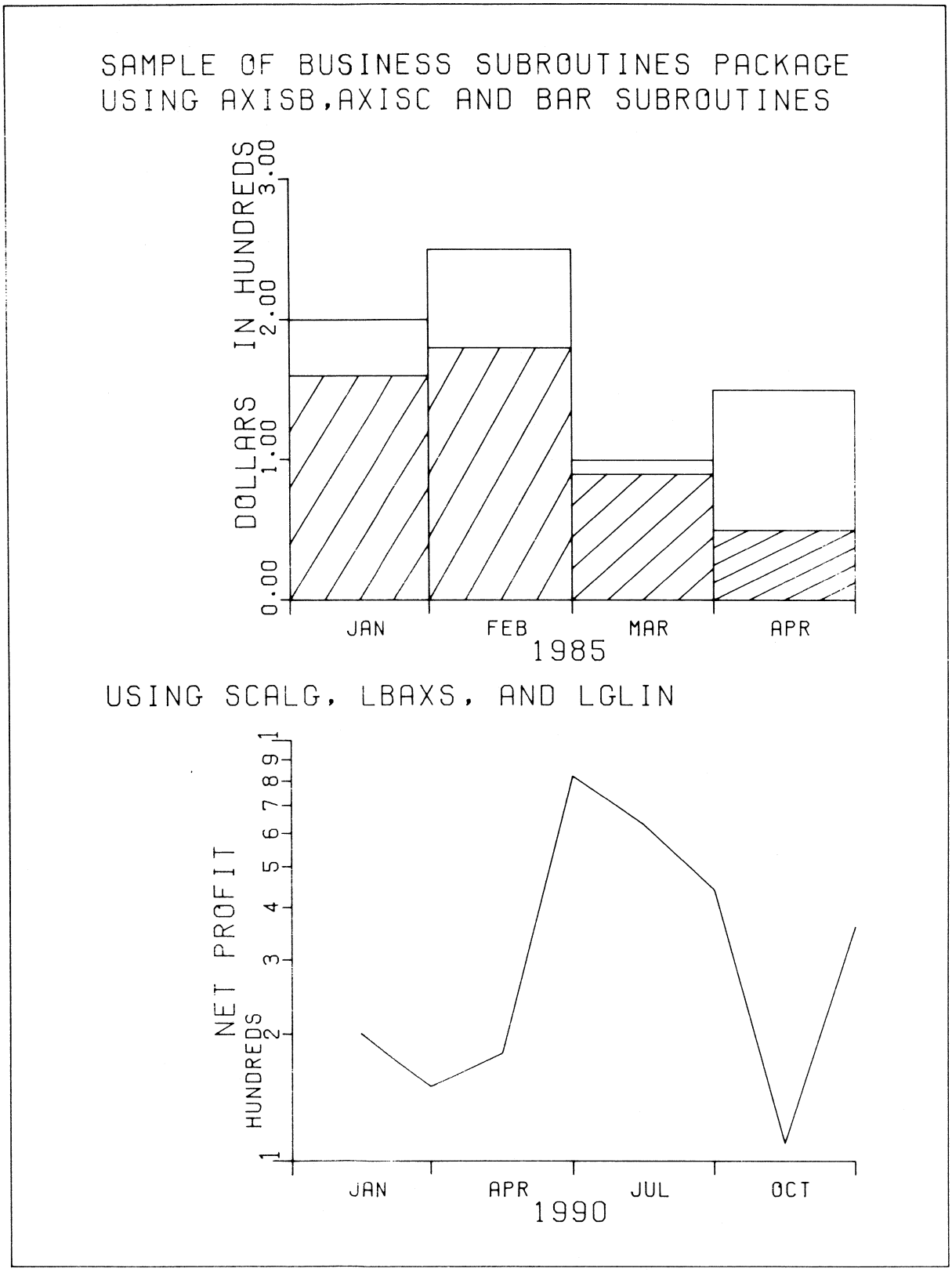


Figure 1-1. Sample of Business Applications Software

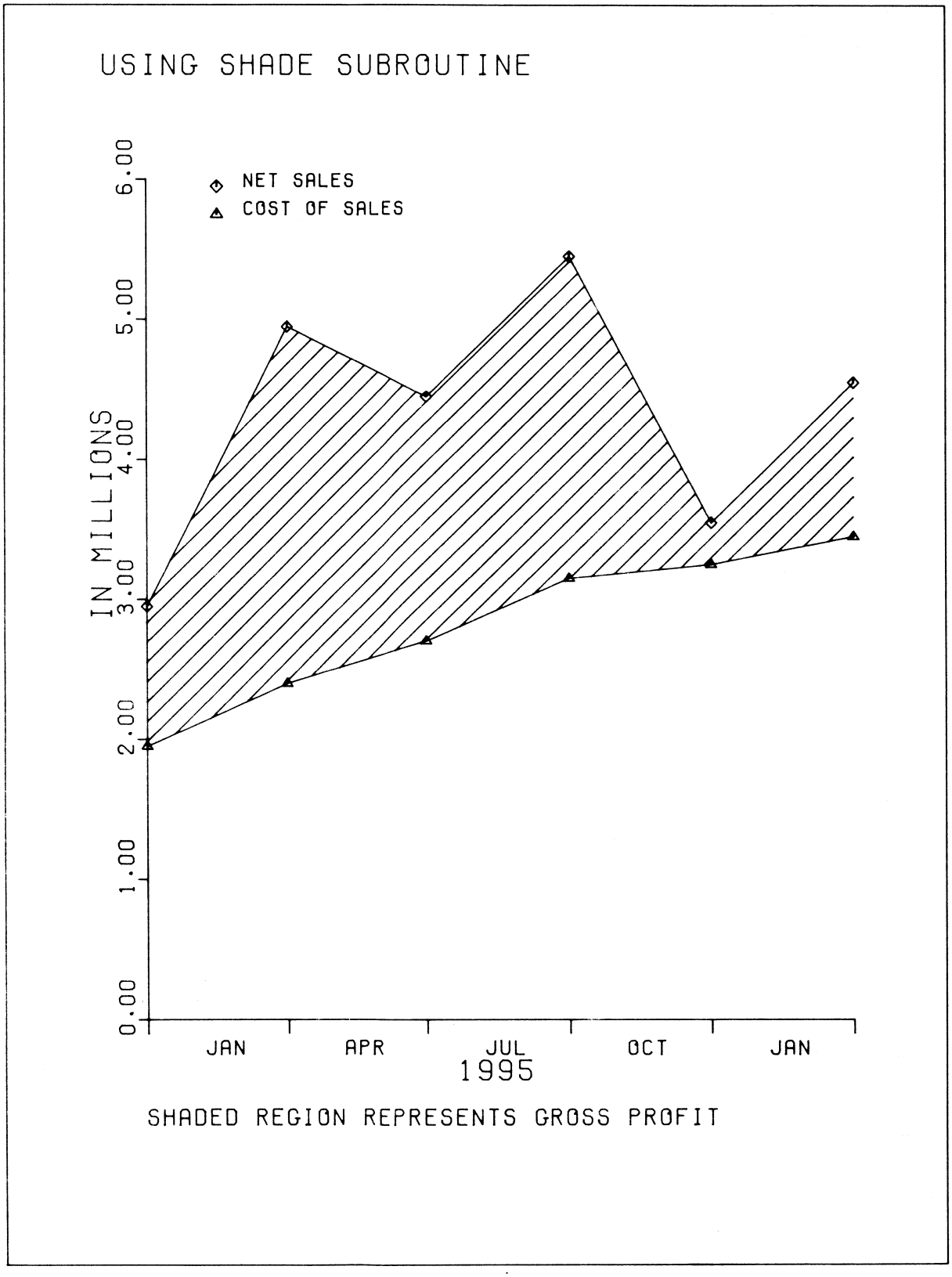


Figure 1-2. Sample of Business Applications Software

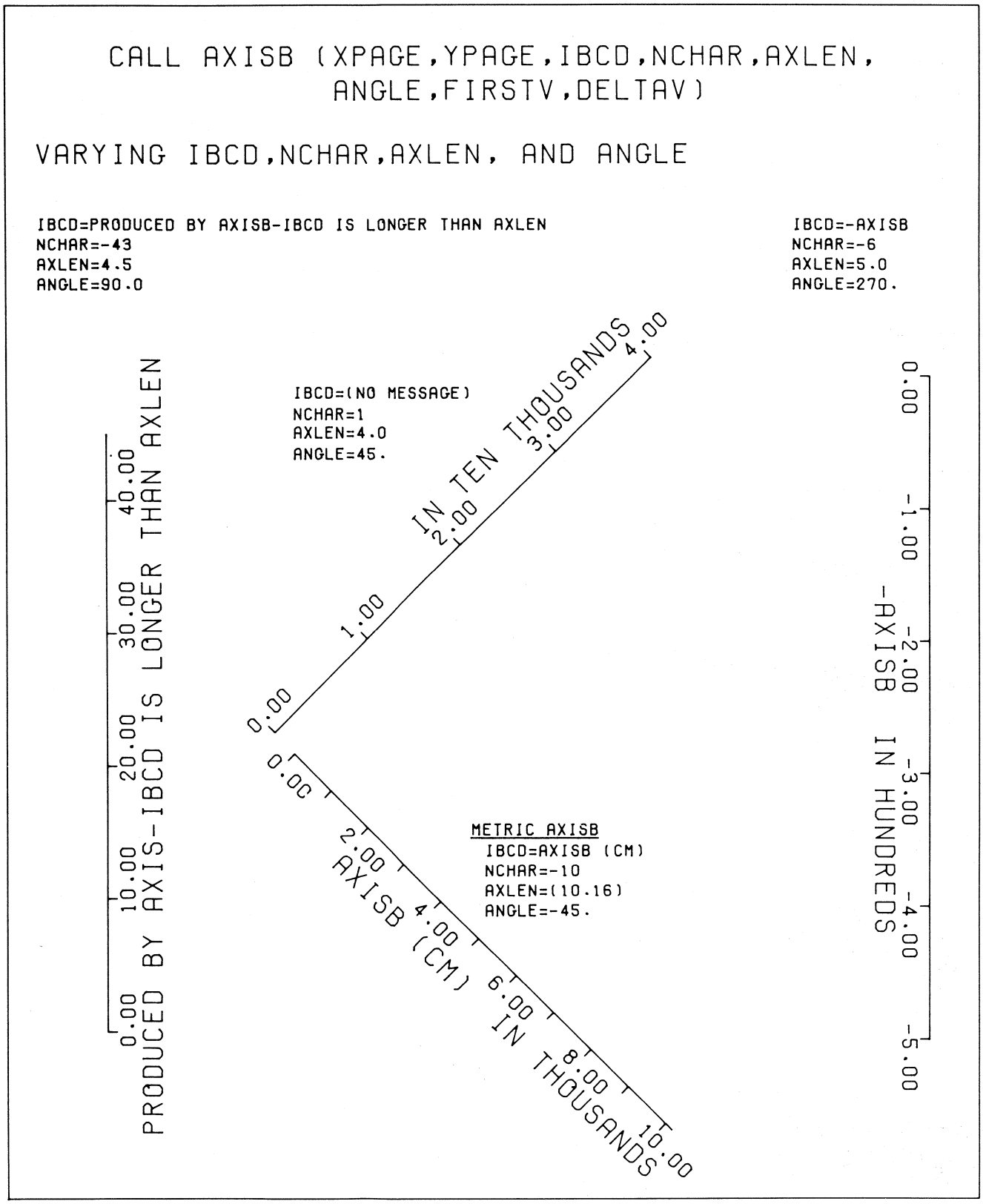


Figure 2-1. Sample of AXISB Subroutine

SECTION 2
AXISB SUBROUTINE

GENERAL

AXISB is a FORTRAN subroutine which draws an axis with business-oriented labeling and a title.

CALLING SEQUENCE

CALL AXISB (XPAGE, YPAGE, IBCD, NCHAR, AXLEN, ANGLE, FIRSTV, DELTAV)

XPAGE, YPAGE	are the coordinates, in inches (centimeters), of the axis starting point.
IBCD	is the alphanumeric array data to be used as the axis title (may be one or more characters).
NCHAR	is the number of characters in the axis title. If NCHAR is:
	negative, the axis title and coordinate labeling are placed on the clockwise side of the axis;
	positive, the axis title and coordinate labeling are placed on the counterclockwise side of the axis.
AXLEN	is the axis length, in inches (centimeters).
ANGLE	is the angle, in degrees, at which the axis is to be drawn. When the angle is positive, the axis is rotated counterclockwise from the horizontal line extending to the right from (XPAGE, YPAGE).
FIRSTV	is the value of annotation at the first tick mark, which falls at (XPAGE, YPAGE).
DELTAV	is the difference in label values from one tick mark to the next tick mark. Tick marks are spaced one inch (centimeter) apart.

Examples of parameter usage are shown in Figures 2-1, 2-2, and 2-3.

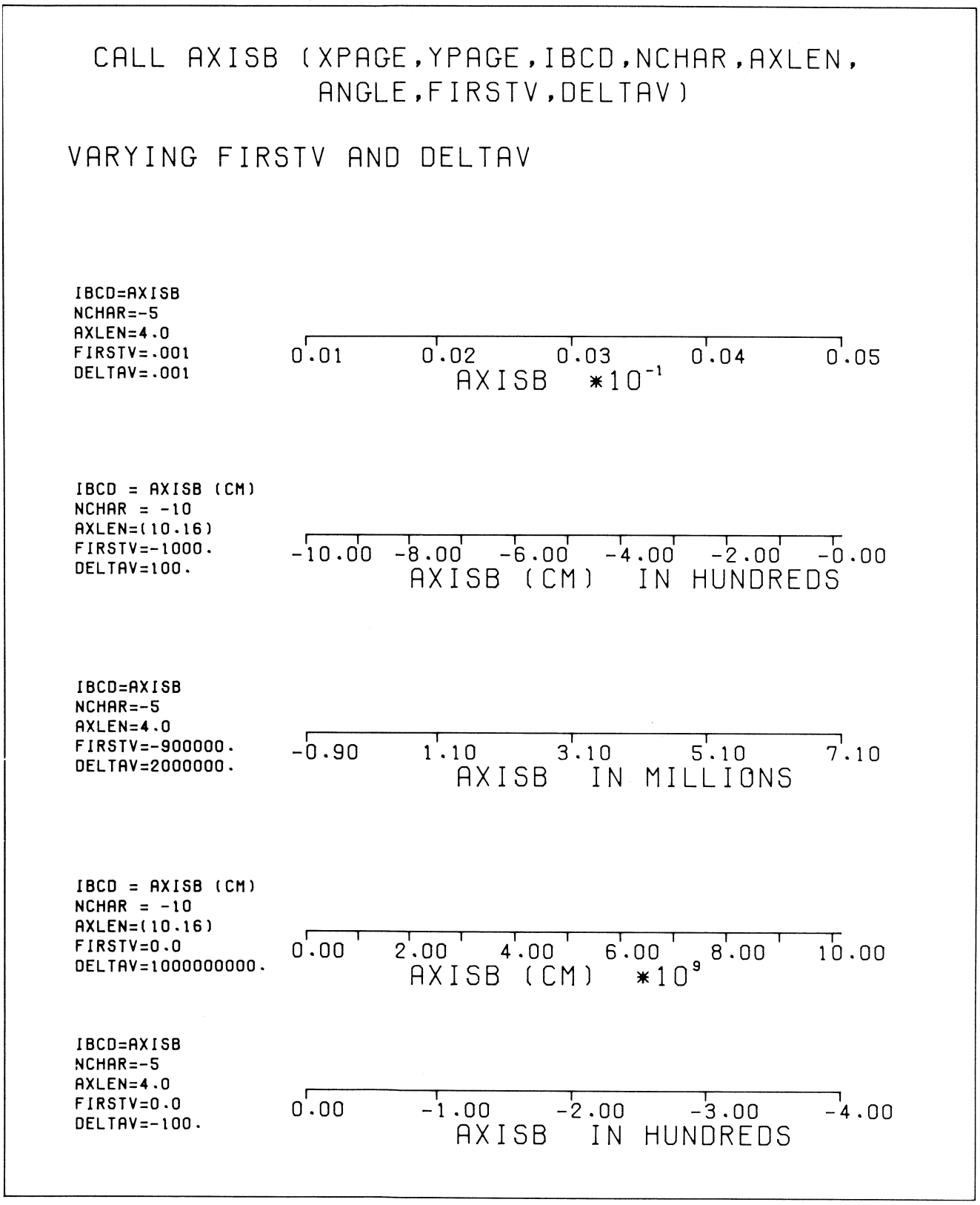


Figure 2-2. Sample of AXISB Subroutine

COMMENTS

The value in DELTAV is scaled so that the adjusted delta value lies between 0.01 and 100. The appropriate scale factor annotation, as shown below, appears after the axis title. N is the integer power of 10 used to bring DELTAV within the allowable range.

VALUE IN DELTAV	VAULE OF ADJUSTED DELTAV	SCALE FACTOR ANNOTATION
DELTAV<0.01	DELTAV * 10^{N}	$*10^{-N}$
0.01 ≤ DELTAV<100	DELTAV	NONE
100 ≤ DELTAV<1,000	DELTAV/100	IN HUNDREDS
1,000 ≤ DELTAV<10,000	DELTAV/1,000	IN THOUSANDS
10,000 ≤ DELTAV<100,000	DELTAV/10,000	IN TEN THOUSANDS
100,000 ≤ DELTAV<1,000,000	DELTAV/100,000	IN HUNDRED THOUSANDS
1,000,000 ≤ DELTAV<10,000,000	DELTAV/1,000,000	IN MILLIONS
DELTAV≥10,000,000	DELTAV/10^{N}	$*10^{N}$

OTHER FUNCTIONAL SUBROUTINES USED

None.

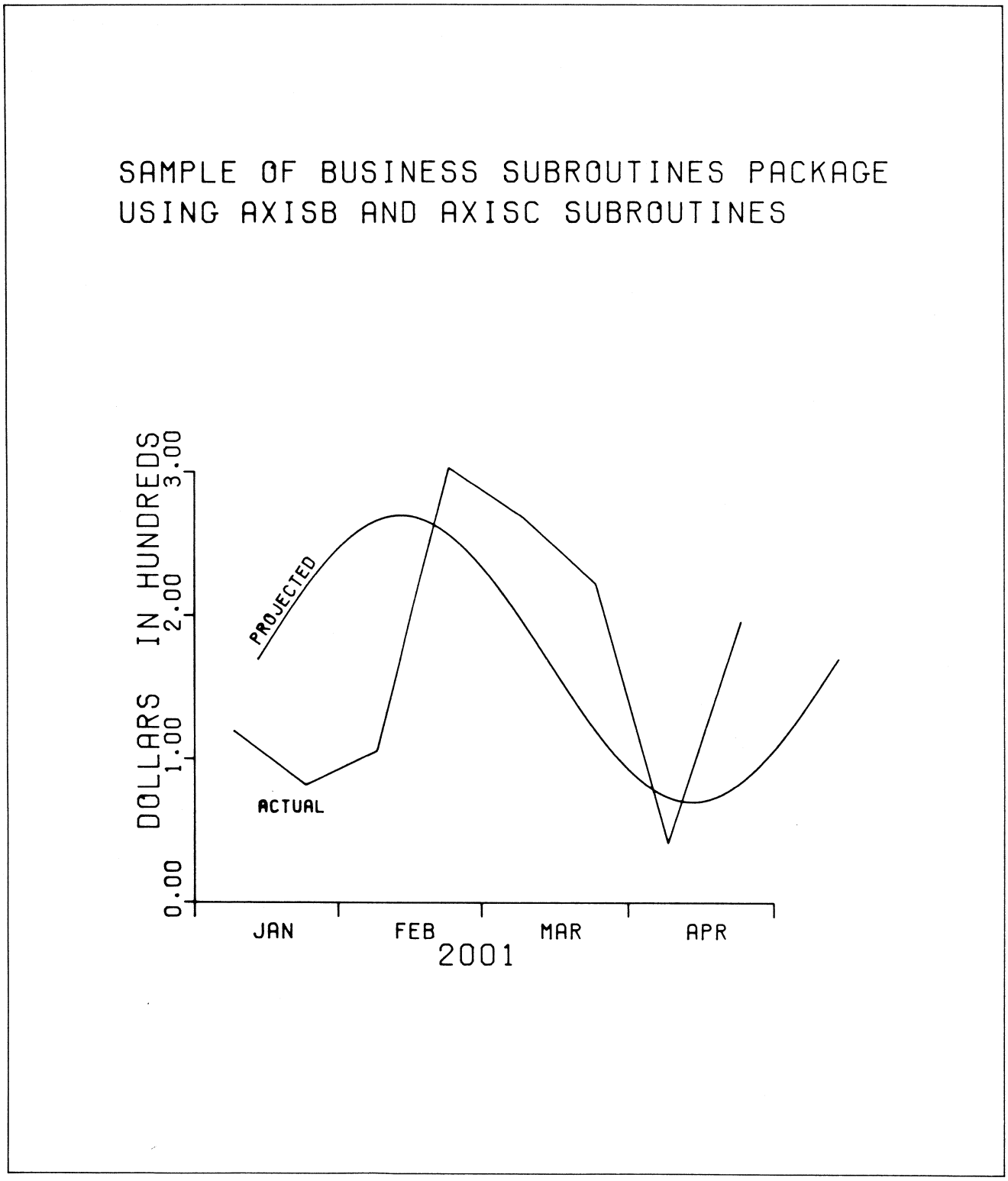


Figure 2-3. Sample of AXISB and AXISC Subroutines

SECTION 3
AXISC SUBROUTINE

AXISC is a FORTRAN subroutine which draws an axis with calendar-month annotation and a title.

CALLING SEQUENCE

CALL AXISC (XPAGE, YPAGE, IBCD, NCHAR, AXLEN, ANGLE, FIRSTV, DELTAV)

XPAGE, YPAGE — are the coordinates, in inches (centimeters), of the axis starting point.

IBCD — is the alphanumeric array data to be used as the axis title (may be one or more characters).

NCHAR — is the number of characters in the axis title. If NCHAR is:

- negative, the axis title and coordinate labeling are placed on the clockwise side of the axis;
- positive, the axis title and coordinate labeling are placed on the counterclockwise side of the axis.

AXLEN — is the axis length, in inches (centimeters).

ANGLE — is the angle, in degrees, at which the axis is to be drawn. When the angle is positive, the axis is rotated counterclockwise from the horizontal line extending to the right from (XPAGE, YPAGE).

FIRSTV — is a value representing the month of the year, to establish annotation at the first tick mark, which falls at (XPAGE, YPAGE). Only the integer portion is used; fractional values are ignored.

VALUE	STARTING MONTH
1.0	January
2.0	February
.	.
.	.
.	.
12.0	December

DELTAV — is the number of months to be labeled between tick marks. Only the integer portion is used; fractional values are ignored. DELTAV must not be negative.

For examples of parameter usage, see Figures 2-3 and 3-1.

OTHER FUNCTIONAL SUBROUTINES USED

None.

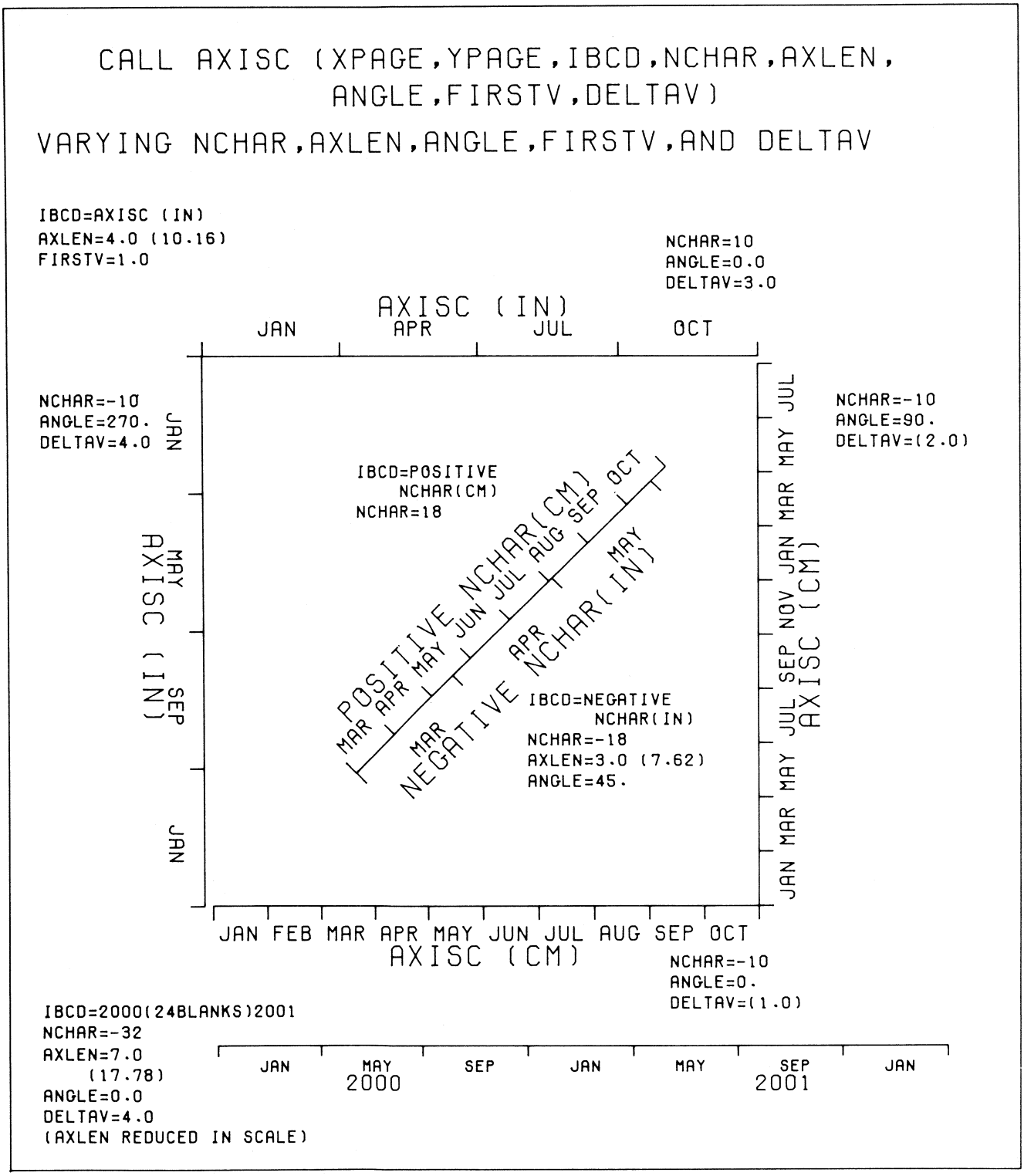


Figure 3-1. Sample of AXISC Subroutine

SECTION 4
BAR SUBROUTINE

GENERAL

BAR is a FORTRAN subroutine which draws bars with or without hatching.

CALLING SEQUENCE

CALL BAR (XPAGE, YPAGE, ANGLE, HEIGHT, WIDTH, SH, IHAT, NPI)

XPAGE, YPAGE — are the lower left coordinates of the bar, in inches (centimeters).

ANGLE — is the angle, in degrees, at which the base of the bar is to be drawn. The bar is rotated about a line extending to the right from XPAGE, YPAGE.

HEIGHT — is the height of the main bar, in inches (centimeters).

WIDTH — is the width of the bar, in inches (centimeters).

SH — is the dimension in inches (centimeters) of the shaded part of the bar, which is hatched according to the IHAT code. SH may equal HEIGHT, if desired, to cause the full bar to be hatched.

SH = Fraction of bar to be shaded * HEİGHT

IHAT — is the hatching code. The hatching produced is parallel to a diagonal of the shaded bar.

- IHAT = 1 — Draw bar only, no hatch lines.
- = 2 — The diagonal extends from the lower left to the upper right corner.
- = 3 — The diagonal extends from the upper left to the lower right corner.
- = 4 — Hatch both ways, resulting in a lattice pattern.

NPI — is the number of lines of hatching per inch (centimeter) along the base. The increment between hatch lines may not be exactly 1/NPI. An adjustment is made so that all hatching is uniform and NPI is not exceeded.

Examples of parameter usage are shown in Figure 4-1.

COMMENTS

The diagonal is always plotted if IHAT $\neq$ 1. To prevent overinking and other problems of clarity, hatch lines which are nearly coincident are not plotted.

OTHER FUNCTIONAL SUBROUTINES USED

None.

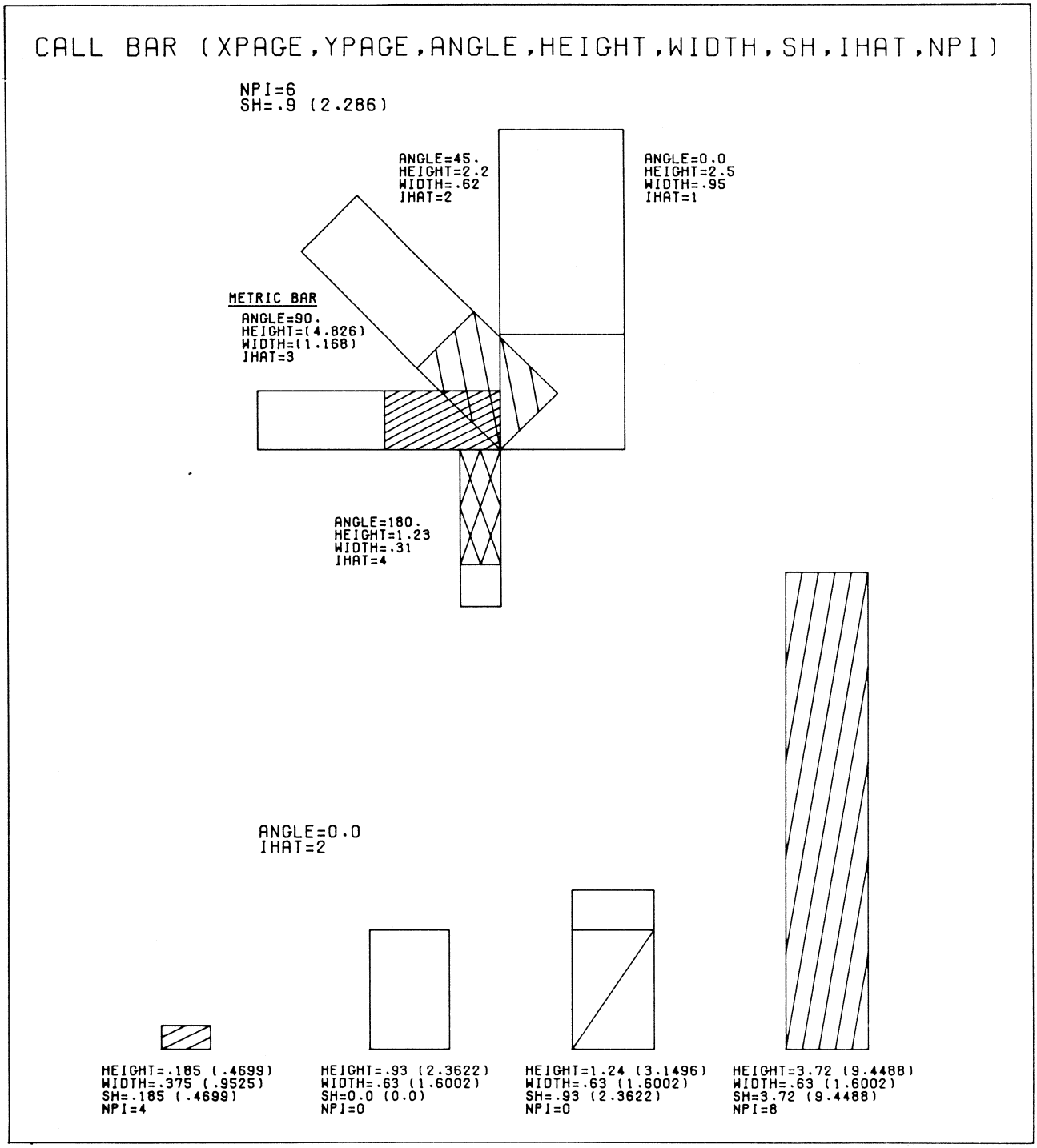


Figure 4-1. Sample of BAR Subroutine

SECTION 5
LBAXS SUBROUTINE

GENERAL

LBAXS is a FORTRAN subroutine which draws a logarithmic axis (in powers of ten) and a title.

CALLING SEQUENCE

CALL LBAXS (XPAGE, YPAGE, IBCD, NCHAR, AXLEN, ANGLE, FIRSTV, DELTAV)

XPAGE, YPAGE	are the coordinates, in inches (centimeters), of the axis starting point.
IBCD	is the alphanumeric array data to be used as the axis title.
NCHAR	is the number of characters in the axis title. If NCHAR is:
	negative, the axis title and coordinate labeling are placed on the clockwise side of the axis;
	positive the axis title and coordinate labeling are placed on the counterclockwise side of the axis.
AXLEN	is the axis length, in inches (centimeters).
ANGLE	is the angle, in degrees, at which the axis is to be drawn. When the angle is positive, the axis is rotated counterclockwise from the horizontal line extending to the right from (XPAGE, YPAGE).
FIRSTV	is the value of coordinate labeling at the beginning of the axis.
DELTAV	is the number of log cycles per inch (centimeter), which is the reciprocal of the length of one cycle in inches (centimeters).

Examples of parameter usage are shown in Figure 5-1.

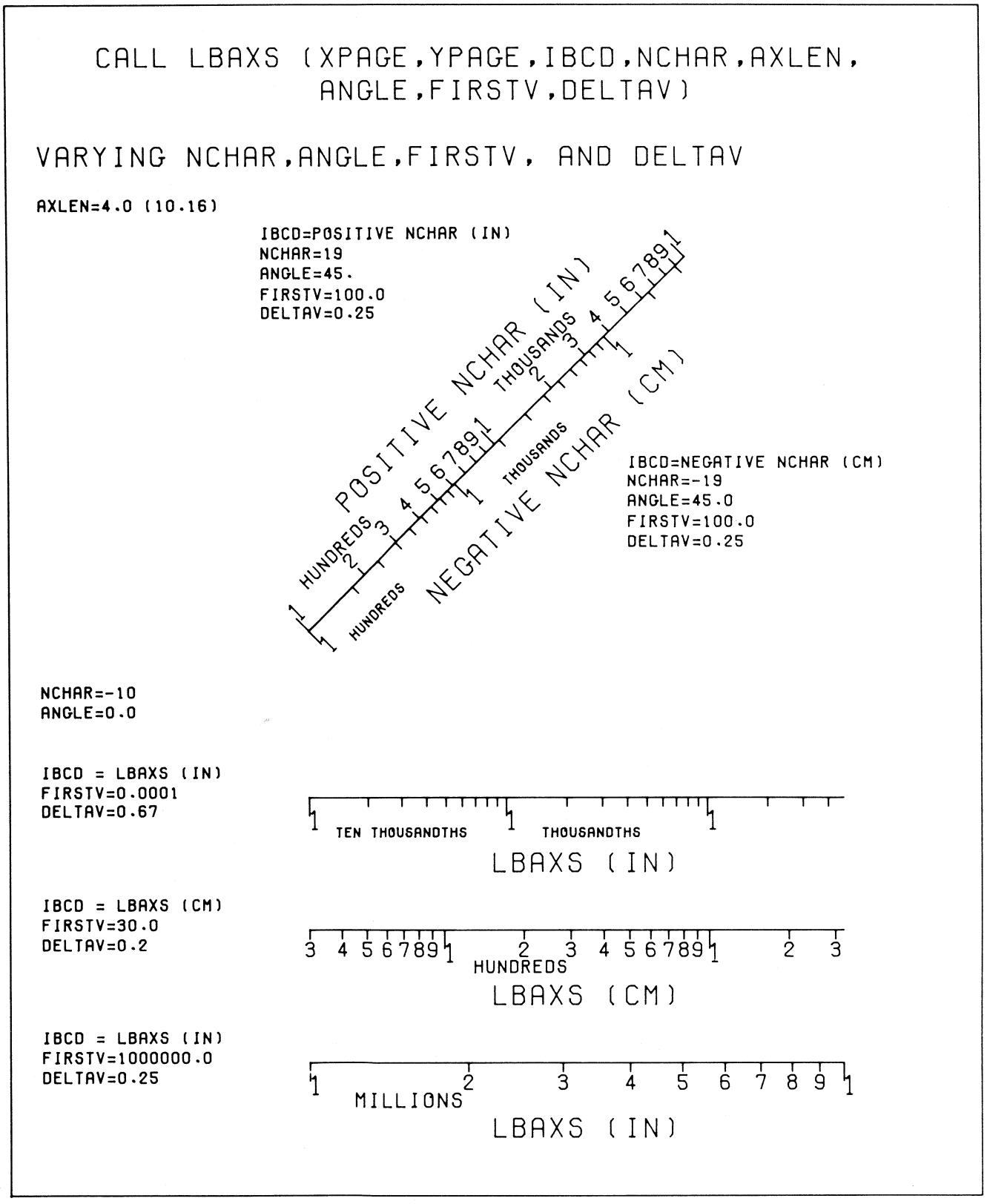


Figure 5-1. Sample of LBAXS Subroutines

COMMENTS

A tick mark is placed along the axis for each power of ten and at each of the nine intermediate integer values.

Annotation is placed at the tick marks as follows:

- If a cycle is not less than two inches (centimeters) long, the integer tick marks are annotated.
- The power-of-ten tick marks are annotated with words (e.g., HUNDREDTHS, BILLIONS), for powers of ten from -14 to +14.
- The power-of-ten tick marks are annotated in the form 10^N for N less than -14 or N greater than +14.
- Since it is controlled by the AXLEN parameter, the axis line may possibly not end at a tick mark.

The SCALG subroutine may be used for determining FIRSTV and DELTAV, if desired.

OTHER FUNCTIONAL SUBROUTINES USED

None.

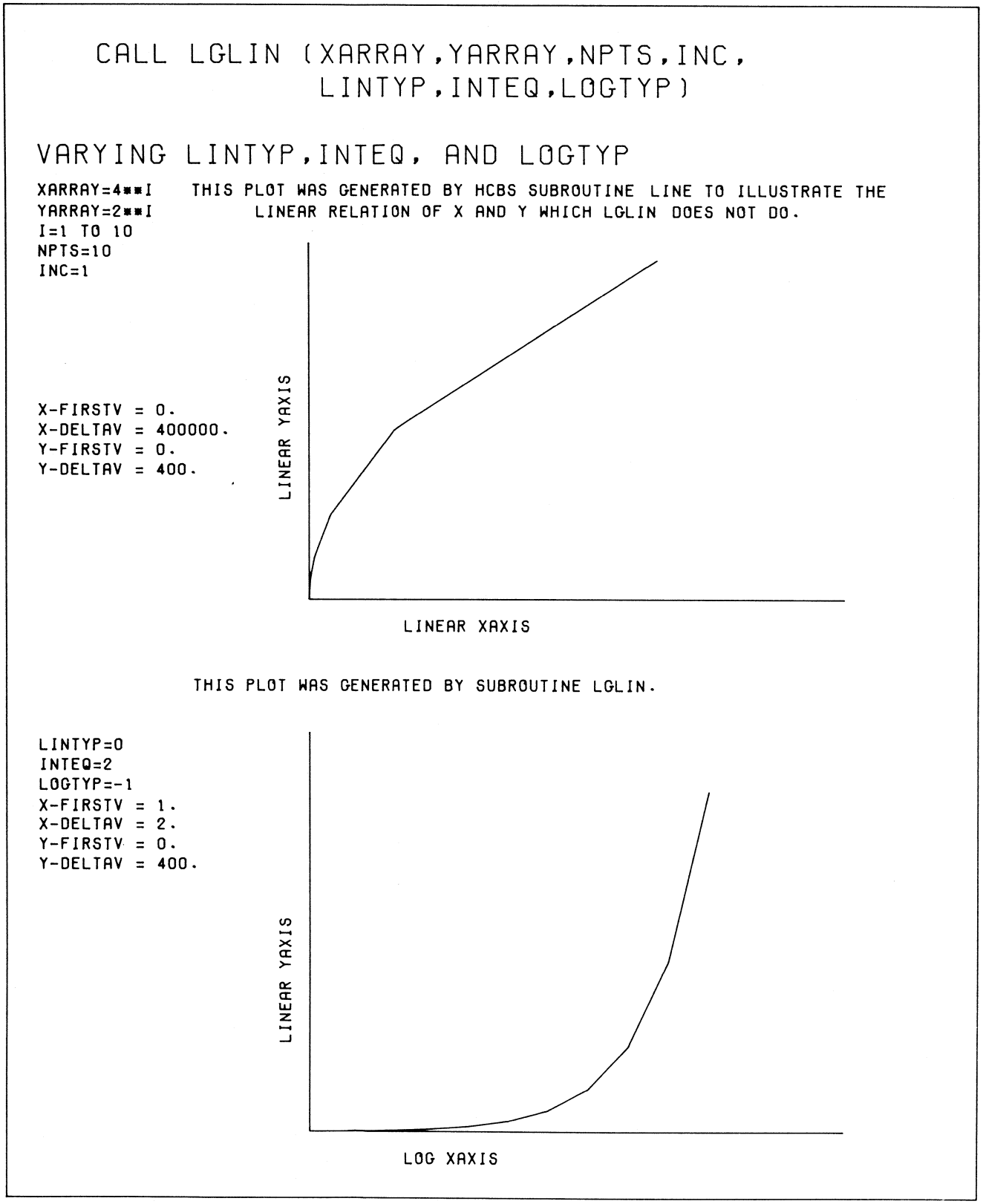


Figure 6-1. Sample of LGLIN Subroutine

SECTION 6
LGLIN SUBROUTINE

GENERAL

LGLIN is a FORTRAN subroutine used to plot data in either log-log or semi-log mode.

CALLING SEQUENCE

CALL LGLIN (XARRAY, YARRAY, NPTS, INC, LINTYP, INTEQ, LOGTYP)

XARRAY, YARRAY — are the arrays containing the variables to be plotted as abscissa and ordinate, respectively. They are either logarithmic or linear, depending on the value of LOGTYP.

NPTS — is the number of points to be plotted.

INC — is the increment between array elements (normally, INC = 1). INC may be greater than 1 if mixed or multidimensioned arrays are used and only every INCth value is to be plotted.

LINTYP — is used to control the type of graph produced:

If LINTYP = 0 a line is plotted between successive data points and no symbol is produced.

= 1 a line plot is produced, with a symbol at each data point.

= n a line plot is produced, with a symbol at every nth data point, starting with the first point.
EXAMPLE: LINTYP = 3, a symbol is plotted at every third point.

= -n connecting lines are not plotted between data points. A symbol appears at every nth data point, starting with the first point.

INTEQ — is the integer equivalent used to specify the symbol to be plotted at a data point. For possible values of INTEQ, refer to Programming CalComp Electromechanical Plotters, (Manual 1006) in the description of the special call to SYMBOL.

INTEQ has no effect when LINTYP = 0.

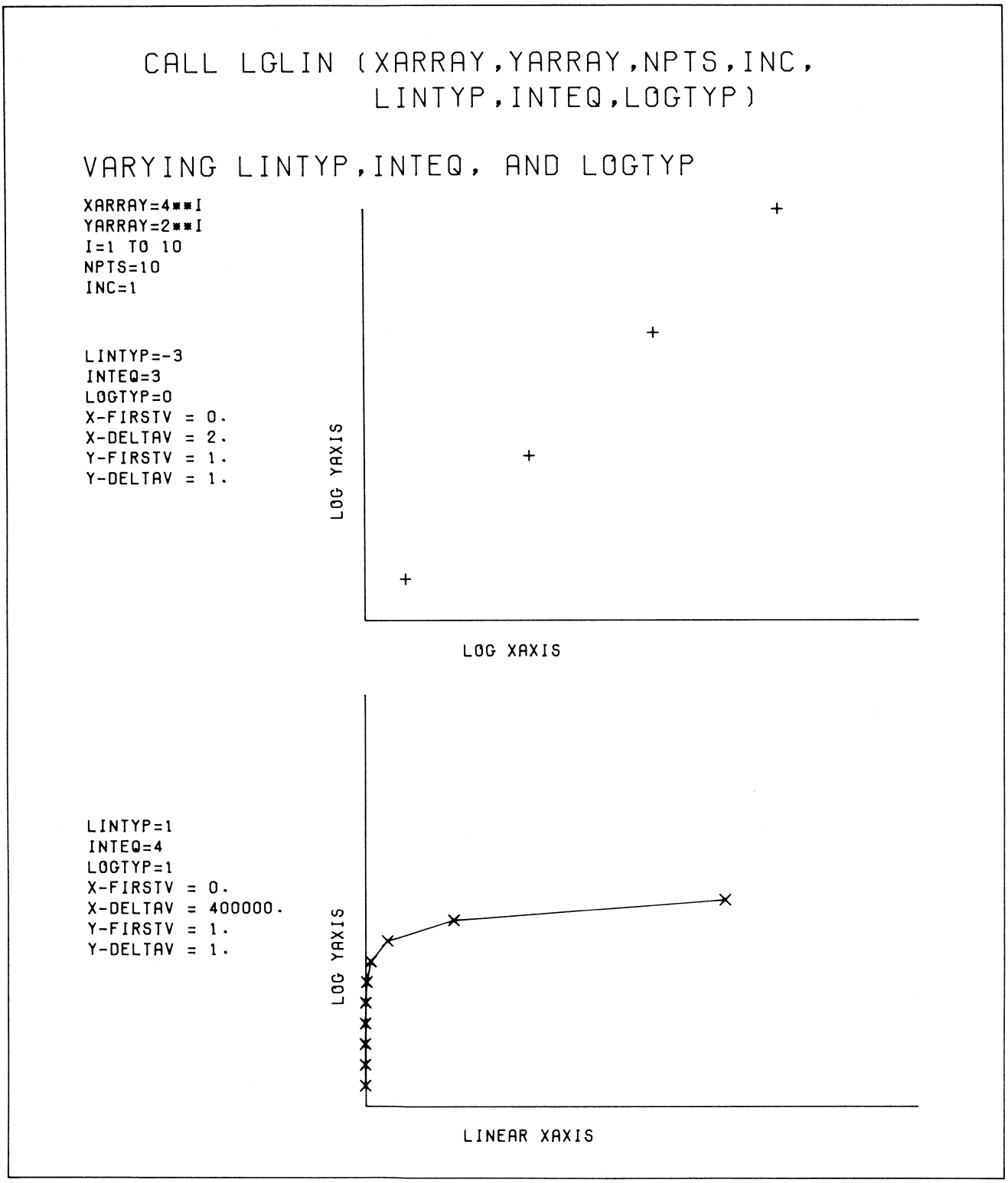


Figure 6-2. Sample of LGLIN Subroutine

CALLING SEQUENCE (cont)

LOGTYP — is a code specifying the type of plot.

If LOGTYP = -1 a plot which is logarithmic in X and linear in Y is produced.

= 0 a plot which is logarithmic in X and Y is produced.

= +1 a plot which is linear in X and logarithmic in Y is produced.

For examples of parameter usage, see Figures 6-1 and 6-2.

COMMENTS

The arrays XARRAY and YARRAY must be dimensioned at least to (NPTS * INC + INC + 1) so as to contain the adjusted minimum value (FIRSTV), and the adjusted delta value (DELTAV) to be used on the respective axis annotation. For the X-array, the adjusted minimum is stored in XARRAY (NPTS * INC + 1) and the adjusted delta is in XARRAY (NPTS * INC + INC + 1). For the Y-array, the minimum is in YARRAY (NPTS * INC + 1), and the delta is in YARRAY (NPTS * INC + INC + 1).

Depending on the value of LOGTYP, FIRSTV and DELTAV may be provided by the SCALE subroutine for a linear axis, or the SCALG subroutine for a logarithmic axis.

If either the XARRAY or YARRAY are to be plotted without translation at a scale factor of 1.0, store 0.0 in FIRSTV and 1.0 in DELTAV.

Unless LINTYP is negative, a line is drawn connecting sequential points. Movement is optimized so that plotting may either begin at the first point and progress forward or begin at the last point and progress backward in the arrays.

OTHER FUNCTIONAL SUBROUTINES USED

None.

SECTION 7
SCALG SUBROUTINE

GENERAL

SCALG is a FORTRAN subroutine used to determine scale factors of the data array to be plotted on a logarithmic scale. The scale factors are those used by subroutines LGLIN and LBAXS.

CALLING SEQUENCE

CALL SCALG (ARRAY, AXLEN, NPTS, INC)

ARRAY	is the array containing data to be scaled. As output, an adjusted minimum value is stored in ARRAY (NPTS * INC + 1), and a delta value, log cycles per inch (centimeters) is stored in ARRAY (NPTS * INC + INC + 1). ARRAY must be dimensioned at least (NPTS * INC + INC + 1) to provide storage for these scale factors.
AXLEN	maximum length over which data is to be plotted, in inches (centimeters).
NPTS	number of ARRAY values to be scaled.
INC	is the increment between array elements (normally, INC = 1). INC may be greater than 1 if mixed or multidimensioned arrays are used and only every INCth value is to be plotted.

COMMENTS

Every INCth element of ARRAY is selected, e.g., a, b, c, d, and e. n is defined as the largest integer such that:

$$10^n \leq \min \{a, b, c, d, e\}$$

m is defined as the smallest integer such that:

$$10^m \geq \max \{a, b, c, d, e\}$$

The adjusted minimum is given as:

$$\text{FIRSTV} = \text{ARRAY (NPTS * INC + 1)} = 10^n$$

The adjusted delta is given as:

$$\text{DELTAV} = \text{ARRAY (NPTS * INC + INC + 1)} = \frac{m-n}{\text{AXLEN}}$$

EXAMPLES

A. For the following array of values

```
ARRAY(1) = 1500.0
ARRAY(2) = 3000.0
ARRAY(3) = 2500.0
ARRAY(4) =  300.0,
```

and the following argument values

```
AXLEN  = 2.0
NPTS   = 4
INC    = 1,
```

the adjusted minimum (FIRSTV) and delta value (DELTAV) stored by SCALG are:

```
FIRSTV:   ARRAY(5) = 100.0
DELTAV:   ARRAY(6) =   1.0.
```

B. If the value of AXLEN in Example A were changed to 4.0, the resultant values stored would be:

```
FIRSTV:   ARRAY(5) = 100.0
DELTAV:   ARRAY(6) =   0.5.
```

C. For the following array of values

```
ARRAY(1) =   1.2*
ARRAY(2) = 100.0
ARRAY(3) =   2.3*
ARRAY(4) =  88.0
ARRAY(5) =   1.8*
ARRAY(6) =   0.0
ARRAY(7) =   0.8*
ARRAY(8) =  20.0
ARRAY(9) =   0.7*
ARRAY(10) =  10.0
```

and the following argument values

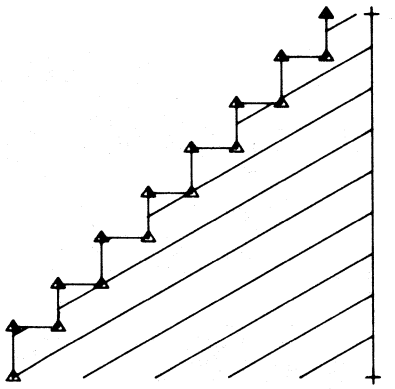
```
AXLEN  = 10.0
NPTS   = 5
INC    = 2,
```

the adjusted minimum FIRSTV) and delta value (DELTAV) are determined from the value of the asterisked items (1, 3, 5, 7, and 9) in the above array. The computed values are also stored two subscript elements apart.

```
FIRSTV:   ARRAY(11) = 0.1
DELTAV:   ARRAY(13) = 0.2
```

```
CALL SHADE(XARAY1,YARAY1,XARAY2,YARAY2,DLIN,
           ANGLE,NPTS1,INC1,NPTS2,INC2)
```

▲ = (XARAY1,YARAY1)
\+ = (XARAY2,YARAY2)
NPTS1=16
NPTS2=2
DLIN=0.2
ANGLE=30.0



XARAY1 = XARAY2 = 0. TO 720.
YARAY1 = SIN(X)
YARAY2 = SIN(X/2)
DLIN=0.1
ANGLE=0.

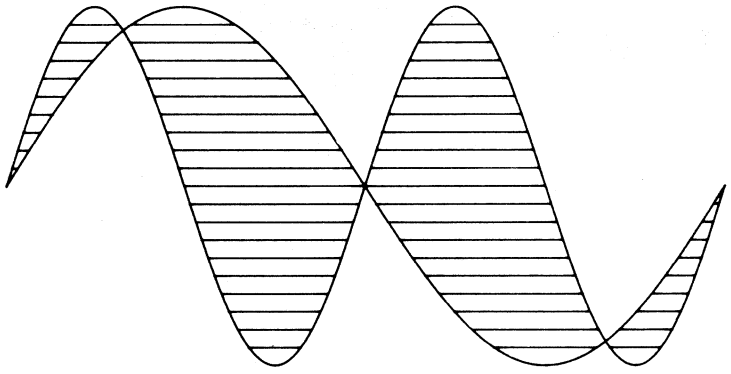


Figure 8-1. Sample of SHADE Subroutine

SECTION 8
SHADE SUBROUTINE

GENERAL

The SHADE subroutine shades any polygon formed by the lines defined by two sets of connected points. SHADE assumes that an imaginary line connects the first points of the two lines and another imaginary line connects the last points of the two lines. Shading is done in the area(s) enclosed by the defined lines and the imaginary lines.

CALLING SEQUENCE

CALL SHADE (XARAY1, YARAY1, XARAY2, YARAY2, DLIN, ANGLE, NPTS1, INC1, NPTS2, INC2)

XARAY1, YARAY1	are the names of the arrays containing the X and Y coordinates of the data points to be plotted for line 1.
XARAY2, YARAY2	are the names of the arrays containing the X and Y coordinates of the data points to be plotted for line 2.
DLIN	is the distance between shading lines.
ANGLE	is the angle (in degrees) the shading lines make with the horizontal X-axis.
NPTS1	is the number of data points forming line 1.
INC1	is the increment between array elements (normally, INC1 = 1). INC1 may be greater than 1 if mixed or multidimensioned arrays are used and only every INC1th value is to be plotted.
NPTS2	is the number of data points forming line 2.
INC2	is the increment between array elements (normally, INC2 = 1). INC2 may be greater than 1, if mixed or multidimensioned arrays are used and only every INC2th value is to be plotted.

For examples of parameter usage, see Figures 8-1 and 8-2.

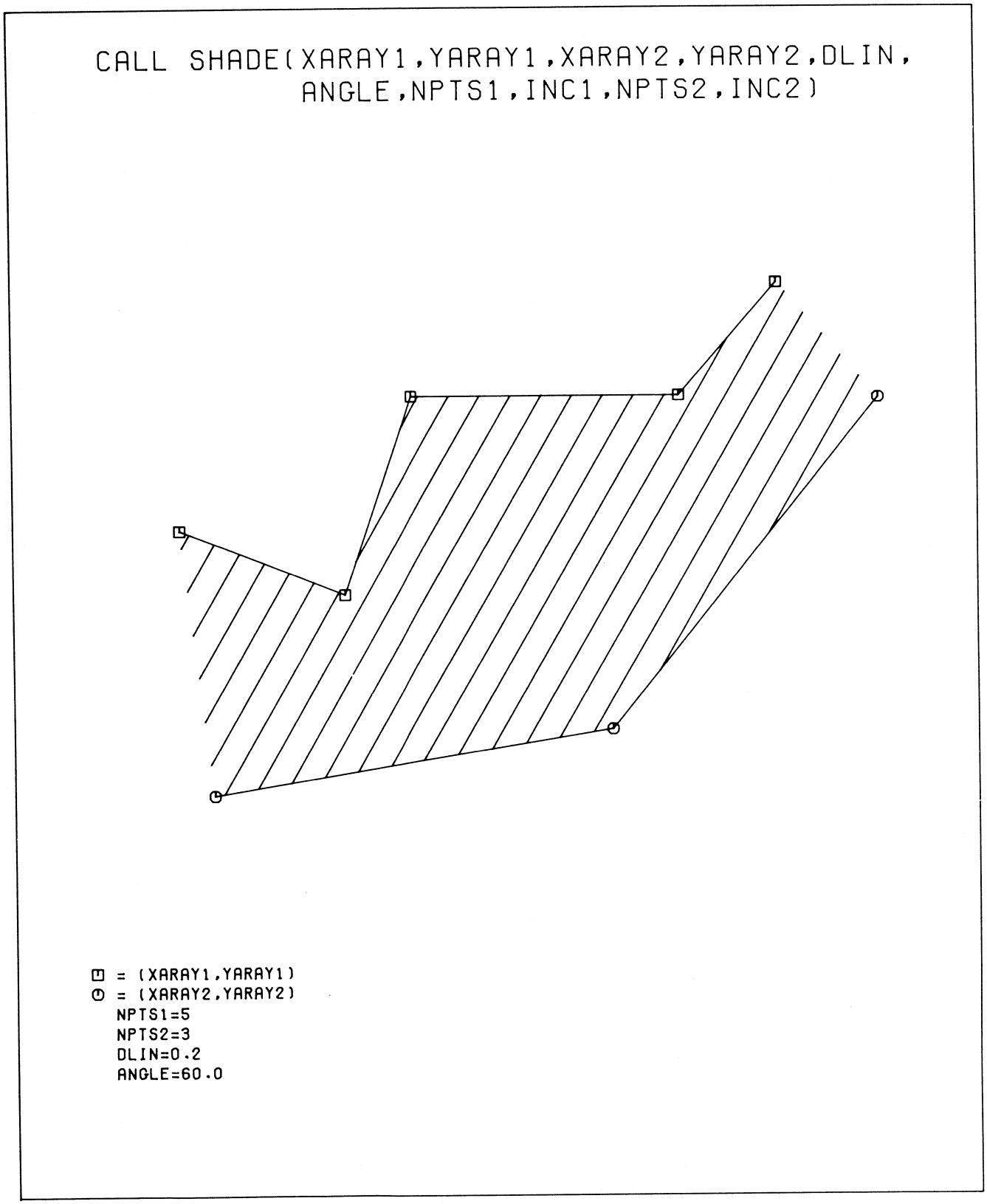


Figure 8-2. Sample of SHADE Subroutine

COMMENTS

The arrays must be dimensioned to at least (NPTS * INC + INC + 1). The adjusted minimum value (FIRSTV) and the adjusted delta value (DELTAV), normally provided by the SCALE subroutine, must be stored following the data in the array.

For each array, the adjusted minimum is stored in element (NPTS * INC + 1) of the array (e.g., XARAY1 (NPTS1 * INC1 + 1)), and the adjusted delta is in element (NPTS * INC + INC + 1) of the array (e.g., XARAY1 (NPTS1 * INC1 + INC1 + 1)).

If the arrays are to be plotted without translation at a scale factor of 1.0, store 0.0 as the minimum and 1.0 as the delta.

OTHER FUNCTIONAL SUBROUTINES USED

STACK

CONTENTS

ILLUSTRATIONS

SECTION 1
INTRODUCTION

GENERAL

This user's manual describes the calling sequence and arguments for the Drafting Applications category of CalComp's Functional Software Library. These five standard FORTRAN subroutines generate calls to CalComp Host Computer Basic Software (HCBS).

Each subroutine description includes plotted-output samples which show the effects of using various arguments values. The argument names used in the calling sequences are arbitrary and need not be used. They have been chosen to illustrate the use of the argument and its particular type, i.e., if the initial of the name is I - N, the argument is an Integer type; otherwise, it is a Real type.

Any other Functional subroutines called by the subroutines described herein are supplied with the category package, but are not intended to be called independently.

FUNCTIONAL SUBROUTINES

The subroutine descriptions are arranged alphabetically, following a composite sample plot (see Figure 1) that illustrates the use of all five subroutines:

AROHD — draws arrowheads

ARROW — draws lines terminated with an arrow

CNTRL — draws center lines

DIMEN — draws annotated dimension lines

LABEL — draws annotation between specified points

NOTE

This manual applies to both inch and metric measurement. Metric parameter values are shown in parentheses in all illustrations. Metric or inch measurement is dependent upon the specific Host Computer Basic Software being used.

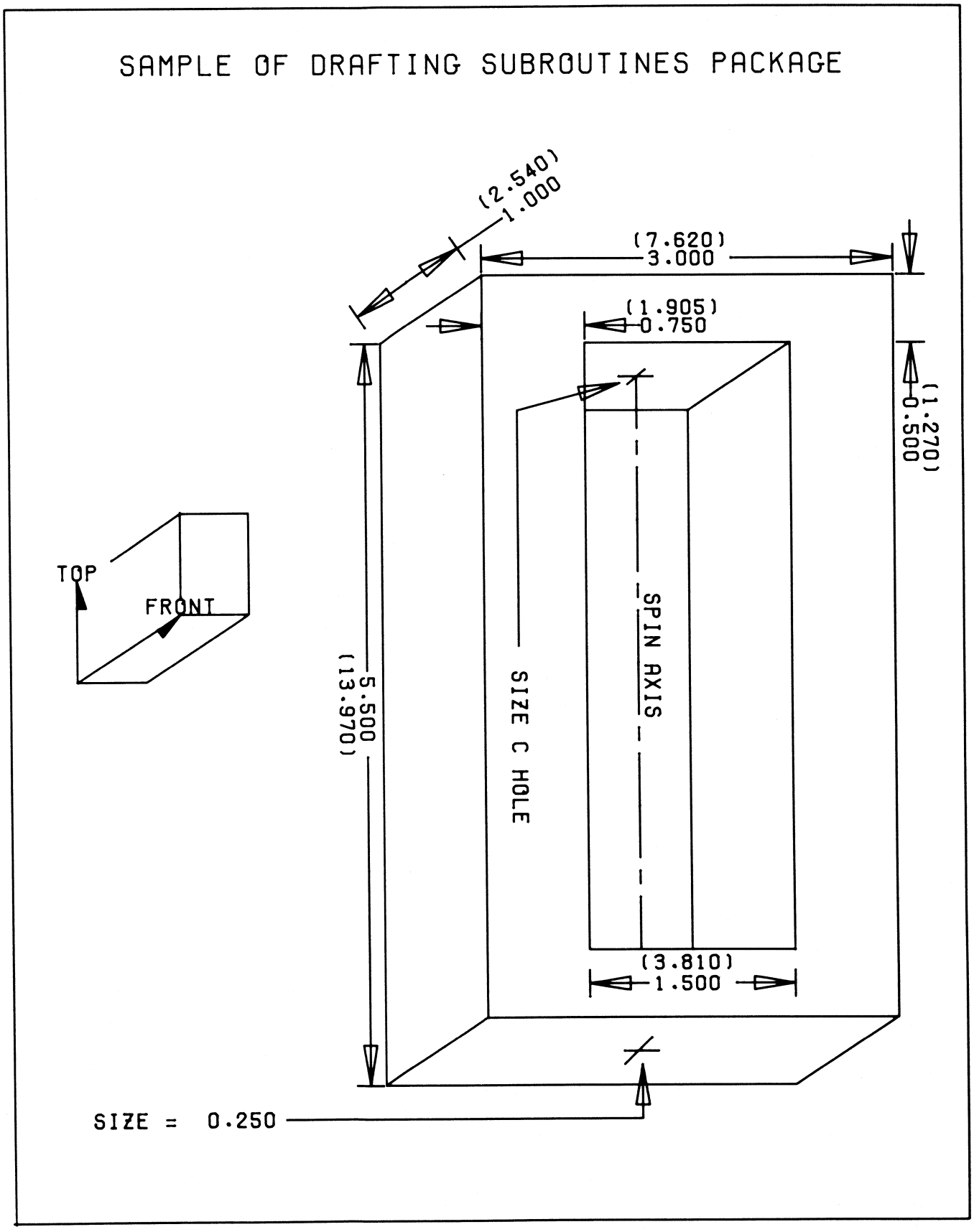


Figure 1. Sample of Drafting Functional Software

SECTION 2
AROHD SUBROUTINE

GENERAL

AROHD is a FORTRAN subroutine which draws an arrowhead at the end of a line segment.

CALLING SEQUENCE

CALL AROHD (XPAGE, YPAGE, XTIP, YTIP, AHLEN, AHWID, ICODE)

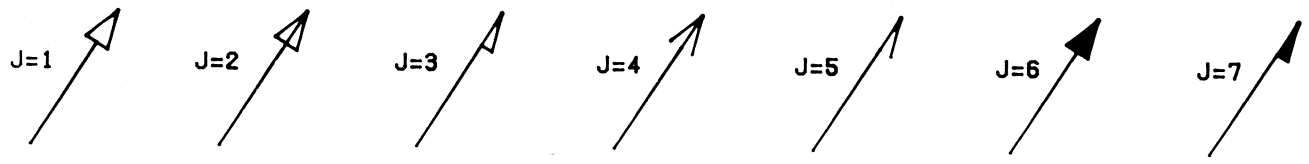
XPAGE, YPAGE	are the coordinates, in inches (centimeters), of the starting point of the line segment and determine the direction of the arrowhead. If ICODE is negative, XPAGE, YPAGE are ignored; and the current pen position as determined by a call to WHERE (in HCBS) is used instead.
XTIP, YTIP	are the coordinates, in inches (centimeters), of the tip of the arrowhead.
AHLEN	is the arrowhead's length, in inches (centimeters).
AHWID	is the arrowhead's width, in inches (centimeters). If AHWID equals zero, the width of the arrowhead is 2/3 AHLEN.
ICODE	is a two-digit decimal code | I | J |

If I = 0, no line is produced from XPAGE, YPAGE to the arrowhead at XTIP, YTIP.

I = 1, a line is produced from XPAGE, YPAGE to the arrowhead at XTIP, YTIP.

I = 2, a line is produced from XPAGE, YPAGE to the arrowhead at XTIP, YTIP and a second arrowhead is also produced at XPAGE, YPAGE.

If J = 1-7, the type of arrowhead produced is as shown below:



OTHER FUNCTIONAL SUBROUTINES USED

None.

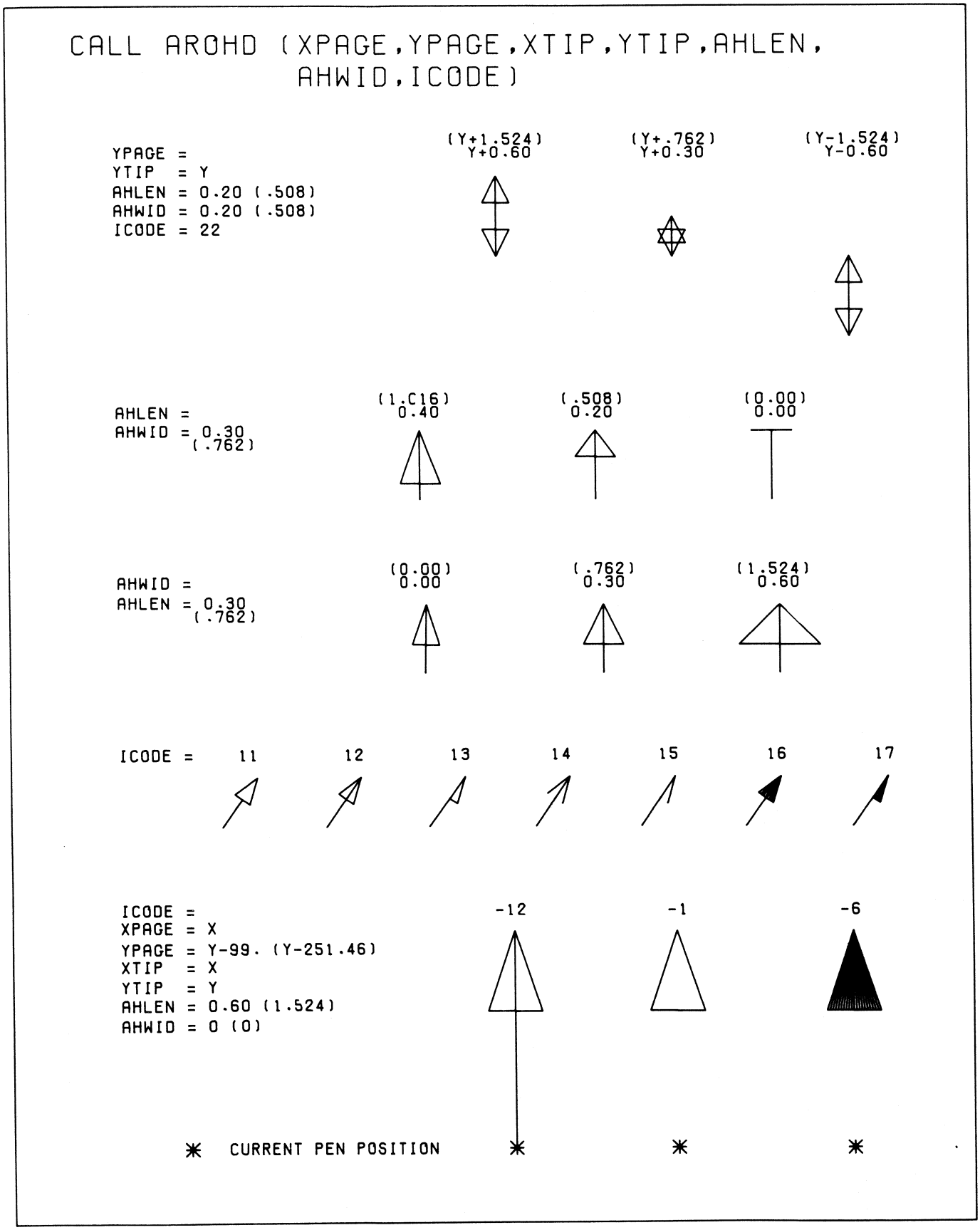


Figure 2. Sample of AROHD Subroutine

SECTION 3
ARROW SUBROUTINE

GENERAL

ARROW is a FORTRAN subroutine which draws a line through an array of data points and places an arrow on the end of the line.

CALLING SEQUENCE

CALL ARROW (XARRAY, YARRAY, NPTS, INC, IATYPE)

XARRAY, YARRAY	are the names of the arrays containing the abscissas and ordinates of the data points to be plotted.
NPTS	is the number of data points to be plotted.
INC	is the increment between array elements (normally, INC =1). INC may be greater than 1 if mixed or multi-dimensioned arrays are used and only every INCth value is to be plotted.
IATYPE	specifies the type of arrow produced (see Figure 3).
	If IATYPE = 1-2, a partial arrowhead is produced: = 3, a full arrowhead is produced; = 4-5, a full arrowhead is produced terminating at extension line(s) perpendicular to the arrow.

COMMENTS

The tip of the arrow appears at the point (XARRAY (NPTS * INC - INC + 1), YARRAY (NPTS * INC - INC + 1).

The arrays XARRAY and YARRAY must be dimensioned with at least (NPTS + 7) * INC elements. The last five elements in the arrays are used by the ARROW subroutine.

The adjusted minimum value (FIRSTV) and the adjusted delta value (DELTAV) must be stored in the data array. These may be provided by the Host Computer Basic Software (HCBS) SCALE subroutine and are stored in the data array following the data.

For the X-array, the adjusted minimum is stored in XARRAY (NPTS * INC + 1), and the adjusted delta is in XARRAY (NPTS * INC + INC + 1). Similarly, for the Y-array, the minimum is in YARRAY (NPTS * INC + 1), and the delta is in YARRAY (NPTS * INC + INC + 1).

If the SCALE subroutine is not used, the user must place the appropriate adjusted minimum values and the adjusted delta values in the specified elements of the arrays. If the XARRAY or YARRAY are to be plotted without translation at a scale factor of 1.0, store 0.0 in FIRSTV and 1.0 in DELTAV.

OTHER FUNCTIONAL SUBROUTINES USED

None.

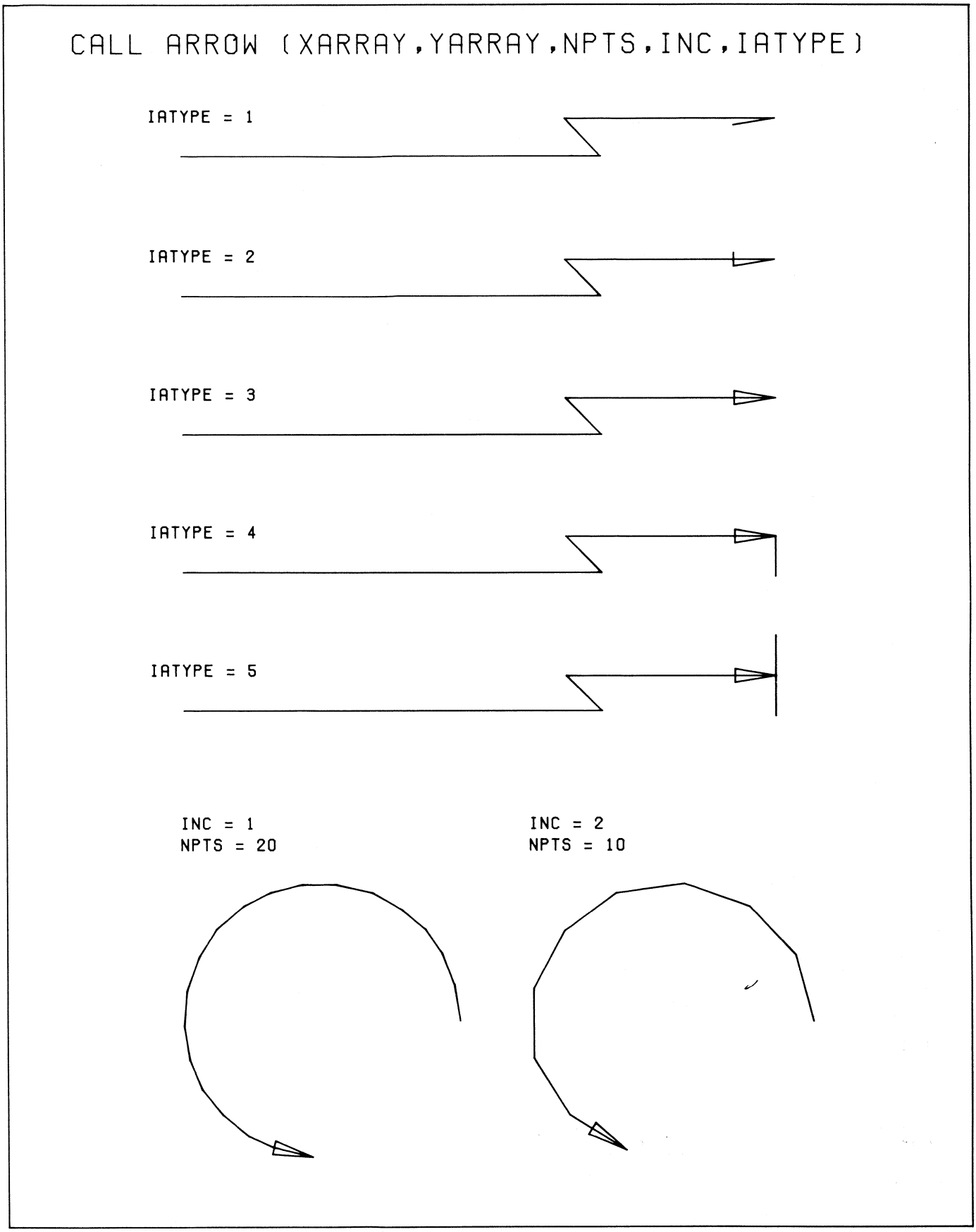


Figure 3. Sample of ARROW Subroutine

SECTION 4
CNTRL SUBROUTINE

GENERAL

CNTRL is a FORTRAN subroutine which draws "center lines" between successive data points. Each "center line" is a broken line consisting of alternating long dashes and short dashes. The long dashes are 4/11 and the short dashes 1/11 of the distance between points.

CALLING SEQUENCE

CALL CNTRL (XARRAY, YARRAY, NPTS, INC)

XARRAY, YARRAY	are the names of the arrays containing the abscissas and ordinates of the data points to be plotted.
NPTS	is the number of data points to be plotted.
INC	is the increment between array elements (normally, INC =1). INC may be greater than 1 if mixed or multi-dimensioned arrays are used and only every INCth value is to be plotted.

COMMENTS

The arrays XARRAY and YARRAY must be dimensioned with at least (NPTS + 2) * INC elements.

The adjusted minimum value (FIRSTV) and the adjusted delta value (DELTAV) must be stored in the data array. These values may be provided by the Host Computer Basic Software (HCBS) SCALE subroutine and are stored in the data array following the data.

For the X-array, the adjusted minimum is stored in XARRAY (NPTS * INC + 1), and the adjusted delta is in XARRAY (NPTS * INC + INC + 1). Similarly, for the Y-array, the minimum is in YARRAY (NPTS * INC + 1), and the delta is in YARRAY (NPTS * INC + INC + 1).

If the SCALE subroutine is not used, the user must place the appropriate adjusted minimum values and the adjusted delta values in the specified elements of the arrays. For a one-to-one correspondence between array data and plotted data, these values should be 0.0 (minimum) and 1.0 (delta).

OTHER FUNCTIONAL SUBROUTINES USED

None.

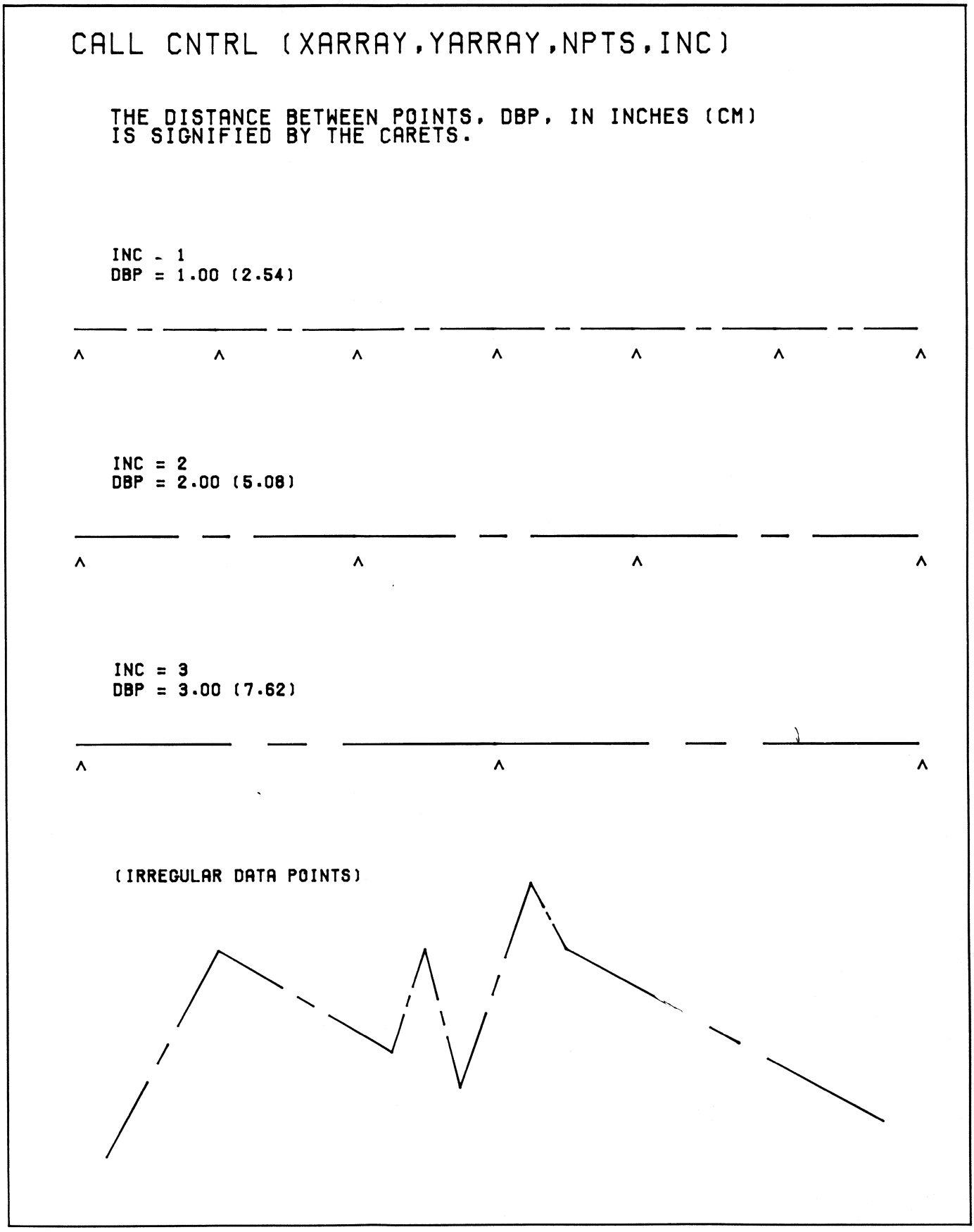


Figure 4. Sample of CNTRL Subroutine

SECTION 5
DIMEN SUBROUTINE

GENERAL

DIMEN is a FORTRAN subroutine which generates a dimension line with short right-angle extension lines at the tips of the line's arrowheads.

CALLING SEQUENCE

CALL DIMEN (XPAGE, YPAGE, DIME, ANGLE, SCALER)

XPAGE, YPAGE	are the coordinates, in inches (centimeters), of the dimension line's starting point.
DIME	is the value to be printed along the dimension line.
ANGLE	is the dimension line's angle, in degrees (positive counterclockwise), from the X-axis.
SCALER	is the scale ratio used in the drawing.

COMMENTS

The actual length of the dimension line is the product of DIME (the labeled length) and SCALER. If the actual length is 1.2 inches (3.0 centimeters) or longer, the annotation is printed in the middle of the dimension line. If the dimension line is between 0.8 (2.0 centimeters) and 1.2 inches (3.0 centimeters) long, the annotation is placed after the line. If the line is less than 0.8 inches (2.0 centimeters) long, the arrowheads are placed outside, along with the annotation.

OTHER FUNCTIONAL SUBROUTINES USED

None.

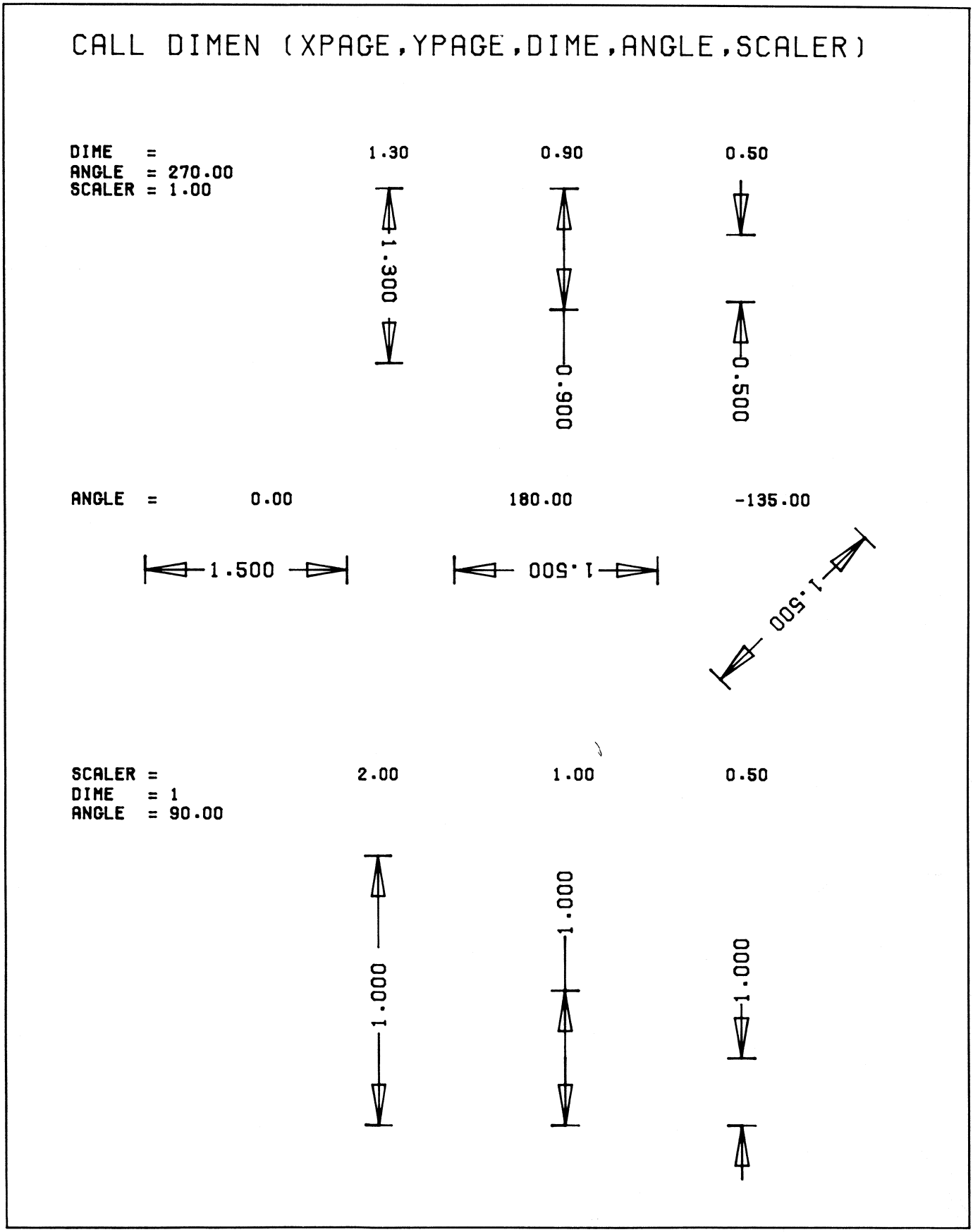


Figure 5. Sample of DIMEN Subroutine

SECTION 6
LABEL SUBROUTINE

GENERAL

LABEL is a FORTRAN subroutine which plots an array of alphameric characters between two points. The characters are automatically centered and their size may be adjusted to fit between the specified points. A floating-point number may be plotted in addition to the alpha characters.

CALLING SEQUENCE

CALL LABEL (XPAGE1,YPAGE1,XPAGE2,YPAGE2,IBCD,NCHAR,HEIGHT,ISIDE,DSTFLN, FPN,NDEC)

XPAGE1,YPAGE1 — are the coordinates, in inches (centimeters), of the starting point, P_1.

XPAGE2,YPAGE2 — are the coordinates, in inches (centimeters), of the ending point, P_2.

IBCD — is the name of the array of characters to be plotted between P_1 and P_2.

NCHAR — is the number of characters from array IBCD to be plotted.

HEIGHT — is the maximum height of the characters, in inches (centimeters). See COMMENTS.

ISIDE — is a two-digit decimal code | I | J |

If I = 0, the IBCD character array is plotted.

I = 1, a floating-point number (FPN) is plotted following IBCD.

If J = 1, the characters are placed on the clockwise side of the vector (viewed looking in the direction of P_1P_2). DSTFLN is measured from vector P_1P_2 to the bottom of the characters.

J = 2, the characters are plotted on the counterclockwise side of the vector (viewed looking in the direction P_1P_2). DSTFLN is measured from vector P_1P_2 to the bottom of the characters.

Whenever ISIDE is negative, array IBCD is inverted for readability. (This rule applies only to vectors that lie in the second or third quadrants.)

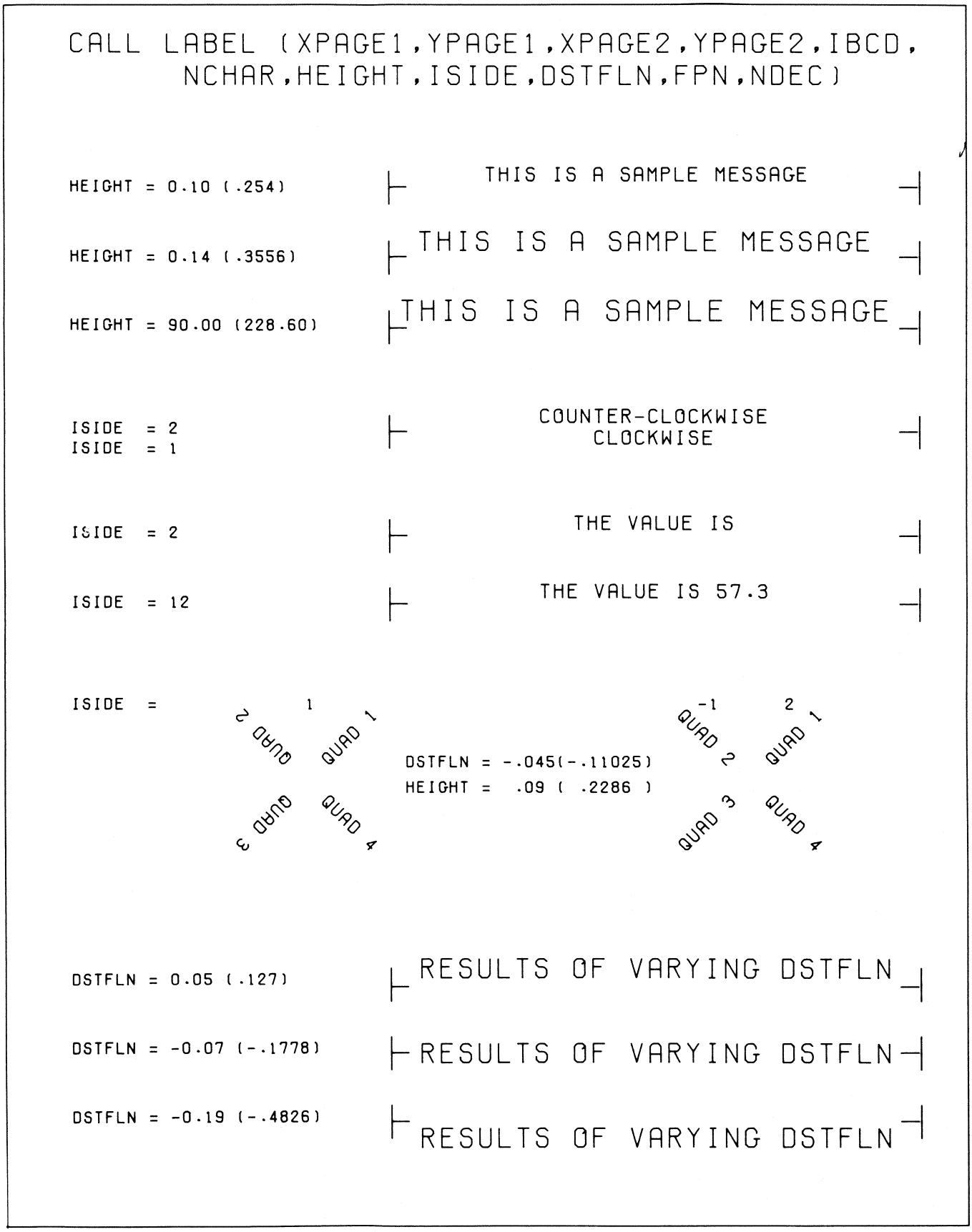


Figure 6. Sample of LABEL Subroutine

CALLING SEQUENCE (cont.)

DSTFLN	is the distance, in inches (centimeters), of IBCD from vector P_1P_2.
FPN	is the floating-point number to be converted and plotted following the IBCD array when the I value of ISIDE is equal to 1.
NDEC	is the number of digits plotted to the right of the decimal point in FPN.

COMMENTS

In centering the characters, both arrays IBCD and FPN are considered.

Six parameters are programmed and compiled at the beginning of the LABEL subroutine code and may have to be changed for specific applications. These parameters are SHR, WHR, THMIN, THMAX, CHMIN, and ENDMA.

SHR and WHR are the space-to-height ratio and width-to-height ratio used by the SYMBOL subroutine called by LABEL. Unless changed, SHR = 1.0 and WHR = 0.57143.

THMIN and THMAX are the angles within which character inversion occurs when ISIDE is negative. Unless changed, THMIN is equal to 93 degrees and THMAX is equal to 267 degrees.

CHMIN is the minimum value to which HEIGHT can be adjusted to make the characters fit within the limits of line P_1P_2. This value is set to 0.07 inches (0.1771 centimeters).

If the characters in array IBCD do not fit between the points P_1P_2, LABEL reduces HEIGHT until the array fits or until HEIGHT = CHMIN. Adjusting HEIGHT causes the width of the IBCD character string to change accordingly. If HEIGHT has to be made less than CHMIN, no characters are plotted.

ENDMA is the minimum margin to be left at each end of the plotted characters. This margin is set to 0.1 inch (0.254 centimeters).

OTHER FUNCTIONAL SUBROUTINES USED

None.

CONTENTS

ILLUSTRATIONS

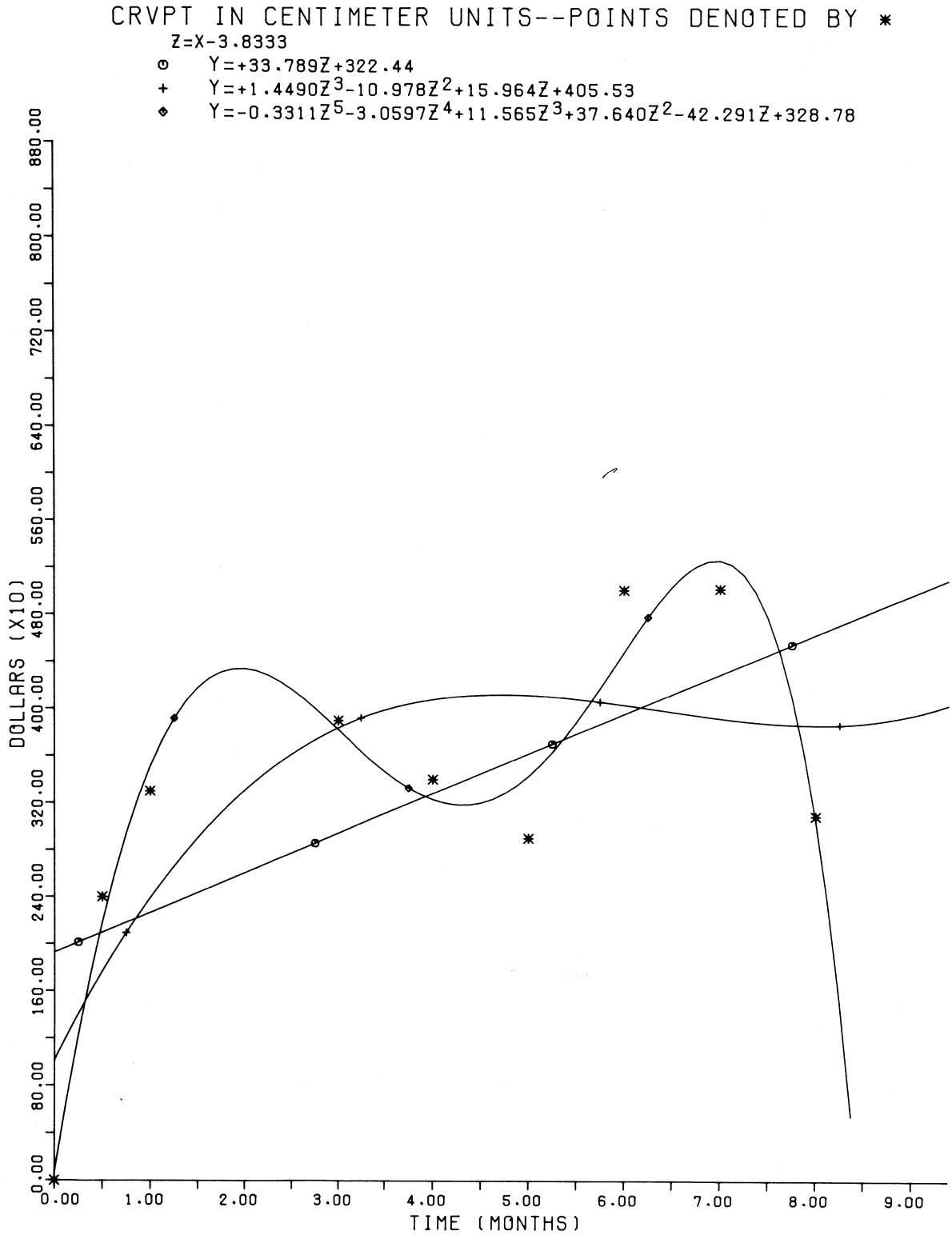


Figure 1-1. Sample of Metric CRVPT Subroutine

SECTION 1
INTRODUCTION

GENERAL

This user's manual describes the calling sequence and arguments for the CRVPT module of CalComp's Functional Software Library. CRVPT and its related subroutines generate calls to CalComp Host Computer Basic Software (HCBS).

Plotted-output samples show the effects of using various argument values. The argument names used in the calling sequences are arbitrary and need not be used. They have been chosen to illustrate the use of the argument and its particular type. If the initial of the name is I - N, the argument is an Integer type; otherwise, it is a Real type.

Three subroutines called by CRVPT are supplied with the software package but are not intended to be called independently.

NOTE

This manual applies to both inch and metric measurement. Metric or inch measurement is dependent upon the specific Host Computer Basic Software being used.

Figure 1-1 is an example of CRVPT output using metric data.

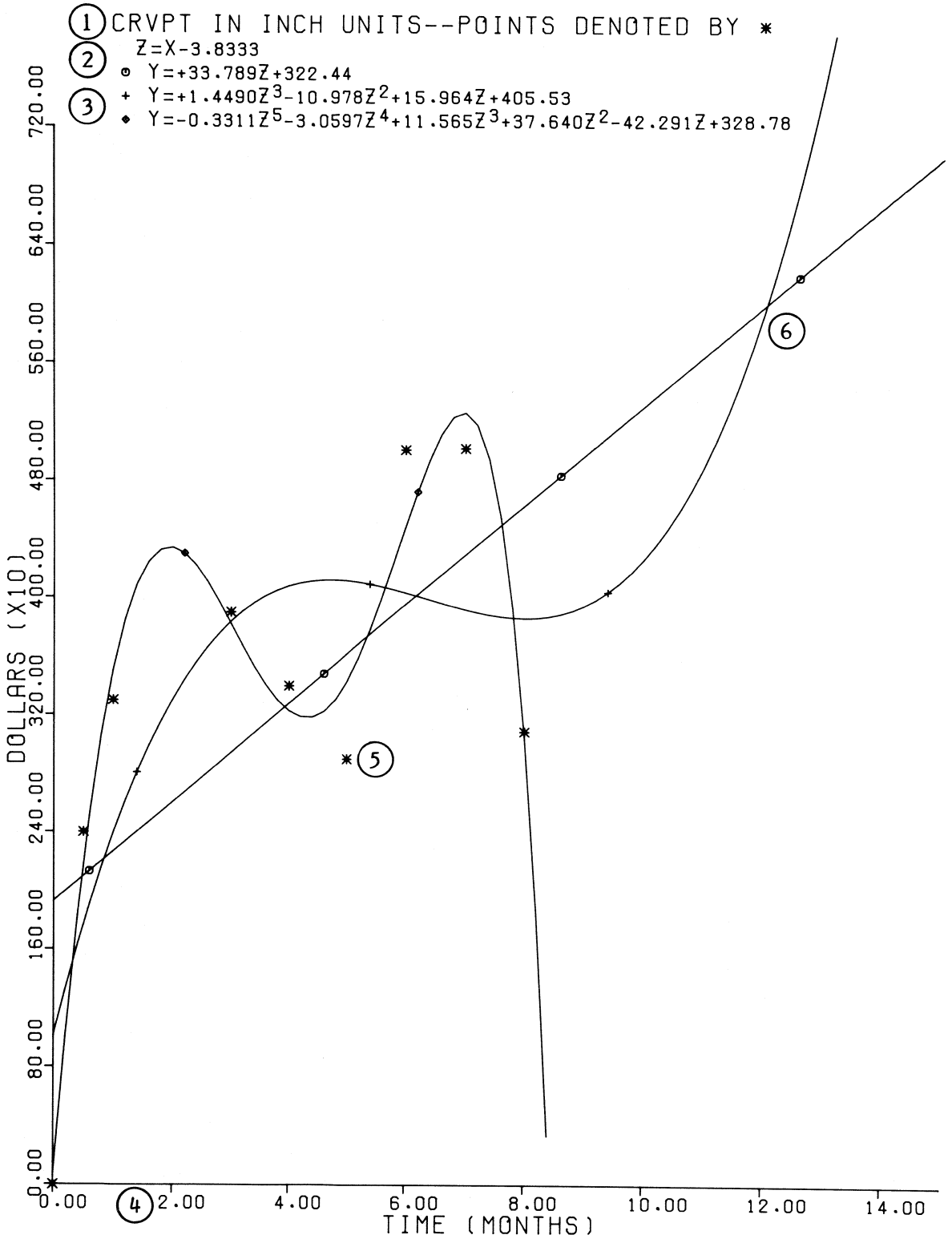


Figure 2-1. Sample of Inch CRVPT Subroutine

SECTION 2
PROGRAM CAPABILITIES

GENERAL

CRVPT is a FORTRAN subroutine which fits a least squares polynomial curve to a set of data points. The fitted curve and data are then plotted, along with reference axes and the equation of the curve.

Figure 2-1 is an example of CRVPT output using inch data.

OUTPUT PLOT FORMAT

The plotted output has the components listed below. Each of the circled numbers corresponds to the same circled numbers in Figure 2-1.

(1) The plot title:

- The title is 0.14 inch (0.3556 centimeters) in height.
- The user specifies any combination of characters to compose the title.
- The title is positioned starting at inch coordinates (X = 0.5, Y = maximum height - 0.2) or centimeter coordinates (X = 1.27, Y = maximum height - 0.508).

(2) Equation of the average X-value:

- The equation is positioned immediately below the title.
- It takes the form: Z = X- (average value of X-array).

(3) Equations for the polynomials of each degree specified on input:

- Each equation is 0.12 inch (0.3048 centimeters) in height.
- The equation is positioned starting at inch coordinates (X = 0.6, Y = maximum height - 0.4) or centimeter coordinates (X = 1.524, Y = maximum height - 1.016).
- Each succeeding polynomial is positioned at a Y-level of 0.2 inch (0.508 centimeters) below the previous polynomial.
- The centered plot symbol corresponding to the degree of the polynomial is displayed adjacent to the equation.

(4) Coordinate axes:

- The user specifies annotation for each axis.
- Axis labeling starts at (X = 0.0, Y = 0.0).
- The user specifies the axis length and coordinate increments.

(5) Data points:

- The coordinates specified by the user are displayed.
- The user selects a centered symbol to display at the points.
- Optionally, a line may be specified to connect the points.

(6) A curve for each requested polynomial:

- The curve is determined by a least-squares approximation.
- The specified plot symbol appears at X-intervals of two inches (5.08 centimeters); the degree of the polynomial is used as the integer code to determine the symbol.

Error messages, when needed:

- The message is 0.2 inch (0.508 centimeters) in height.
- The error text is positioned starting at coordinates (X = maximum width, Y = 0.0).

RESTRICTIONS

- There must be at least one data point.
- The highest allowable degree of polynomial fit is nine.
- The degree of the polynomial to be fit must be at least one less than the number of noncoincident data points.

ERROR MESSAGES

An error message "ERROR NO. N" is displayed at the lower right corner of the plot, if one of the following conditions is found by the program:

If $10 < N < 20$,	the user has requested a polynomial of degree (N-10) but specified less than (N-10) data points.
If $100 < N$	the $(N-100)^{th}$ data point lies outside the range of the plot and does not appear in the display.

If an error is detected, the maximum width is increased by three inches (7.62 centimeters) so that the next plot, if any, will clear the error message. Processing for the current plot is terminated by any error, but the next data case, if any, is read and processed.

SECTION 3
CALLING SEQUENCE

GENERAL

CALL CRVPT (XARRAY,YARRAY,INTEQ,NPTS,INC,SH,SW,IBCDT,
NCHAR,IBCDX,NCHARX,IBCDY,NCHARY,INT)

XARRAY is the array containing the X-coordinates of the data points. XARRAY must be dimensioned to at least NPTS+2 elements.

YARRAY is the array containing the Y-coordinates of the data points. YARRAY must be dimensioned to at least NPTS+2 elements.

Before CRVPT is called, the pen should be positioned at least 0.5 inch (1.27 centimeter) from the bottom of the page to allow for X-axis annotation.

INTEQ is the array of integer codes for the plot symbol associated with each data point, respectively. For possible values of INTEQ, refer to the description of the special call to SYMBOL in Programming CalComp Electromechanical Plotters (Manual 1006).

If INTEQ is negative, it is treated as a single-integer variable, instead of an array, and the plot symbol denoted by the integer equivalent is displayed at each data point.

NPTS is the number of data points. For the first call to CRVPT, NPTS must be greater than zero.

If NPTS is negative, the absolute value of NPTS is used for the number of data points, and the current data set will be plotted with the same scale used for the preceding data set. No new axes or titles will be plotted.

INC is the increment between elements in XARRAY and YARRAY (normally, INC = 1). INC may be greater than 1 if mixed or multidimensioned arrays are used and only every INCth value is to be plotted.

If INC is negative, the absolute value of INC is used for the array increment, and the pen is repositioned at the origin of the present plot before returning to the calling program. This allows the next set of data to be plotted in the same coordinate system.

SH is the maximum height of the plot, in inches (centimeters). The Y-axis will have a length of SH - 1.0 inches (SH - 2.54 centimeters).

SW is the maximum width in inches (centimeters), of the plot (and the X-axis).

IBCDT is the alphameric array containing the plot title.

NCHAR is the number of characters to plot from IBCDT.

IBCDX is the alphameric array containing the X-axis title.

NCHARX is the number of characters to plot from IBCDX.

IBCDY is the alphameric array containing the Y-axis title.

NCHARY is the number of characters to plot from IBCDY.

INT is an array of integers defining the degrees of the polynomials to be calculated and plotted. The last entry must be zero. No entry may be greater than 9.

If the first value of INT is zero, the data points are plotted with connecting lines, and the second value of INT is interrogated for the degree of the first polynomial to be fit.

Any zero INT value following the initial INT value will cause the program to exit from CRVPT.

If INT is negative, it is treated as a single-integer variable. Starting with the units position and progressing to the tens, hundreds, etc., each digit is considered an element of the array and the degree of a polynomial.

OTHER SUBROUTINES USED

CRVFT
FNUM
MATX

SECTION 4
INPUT DATA FORMAT

GENERAL

CRVPT includes a main program to read user input from cards (or disk) which then makes appropriate calls to the CRVPT subroutine. The format of the data for this driver program is given in the following paragraphs.

CARD TYPES

Curve and Frame Card – provides the degree of polynomial curves and frame information.

Title Cards – specify the plot titles.

Data Point Cards – defines the coordinates of the data points in user-specified units.

Curve and Frame Card

The Curve and Frame Card defines the frame dimensions and frame advancing as well as the degree of polynomials to be plotted.

Card Columns	Data Format	Field Name	Description
1-5			(blank)
6	I1	ICODE	Code for specifying the frame advance after plotting the current set of data points. If ICODE = 0 or blank, the data points following the current set are plotted using a new origin, advancing the frame. = 1 the data points following the current set are plotted with the same origin, resulting in no frame advance. In this case, no Title Cards are allowed following the Curve and Frame Card. = 2 the current set is the last set of data cards for this computer run.

Card Columns	Data Format	Field Name	Description
7-16	F10.3	PAGE HEIGHT	Height of the frame, which encloses all titles, axes, and plots.
17-26	F10.3	PAGE WIDTH	Width of the plot.
27			(blank)
28	I1	DEGREE OF POLY (1)	Degree of the first polynomial to be calculated for the set of data points.
29			(blank)
30	I1	DEGREE OF POLY (2)	Degree of the second polynomial (if any) to be calculated for the set of data points.
31			(blank)
32	I1	DEGREE OF POLY (3)	Degree of the third polynomial (if any) to be calculated for the set of data points.
.	.	.	
.	.	.	
.	.	.	
49			(blank)

Card Columns	Data Format	Field Name	Description
50	I1	DEGREE OF POLY (12)	Degree of the twelfth polynomial (if any) to be calculated for the set of data points.
51-80			(not used)

Title Cards

If ICODE = 0 or blank in the Curve and Frame Card, three Title Cards must follow:

- The first card contains the plot title.
- The second card contains the X-axis title.
- The third card contains the Y-axis title.

Card Columns	Data Format	Field Name	Description
1-6	I6	NCHAR	Number of characters in the title.
7-9			(not used)
10-69	A	IBCD	Alphameric title.
70-80			(not used)

Data Point Cards

The least squares approximation of a polynomial is calculated from input points. Each pair of point coordinates is represented by a Data Point Card.

Card Columns	Data Format	Field Name		Description
1-5				(blank)
6	I1	LCODE	= 1	when the card represents the last point of the set.
			= 0	or blank, when there are more points in the set.
7-16	F10.3	XVAL		The value of the X-coordinate of the data point.
17-26	F10.3	YVAL		The value of the Y-coordinate of the data point.
27-28	I2	INTEQ		Enter the integer equivalent, right-justified, of the symbol to be plotted at the coordinates of the data point. For possible values, refer to the description of the special call to SYMBOL in Programming CalComp Electromechanical Plotters (Manual 1006). A blank or zero value of INTEQ indicates the symbol which was used for the previous point. The symbol normally indicated by integer zero must be specified by a value of 99.
29-80				(not used)

CONTENTS

ILLUSTRATIONS

TABLES

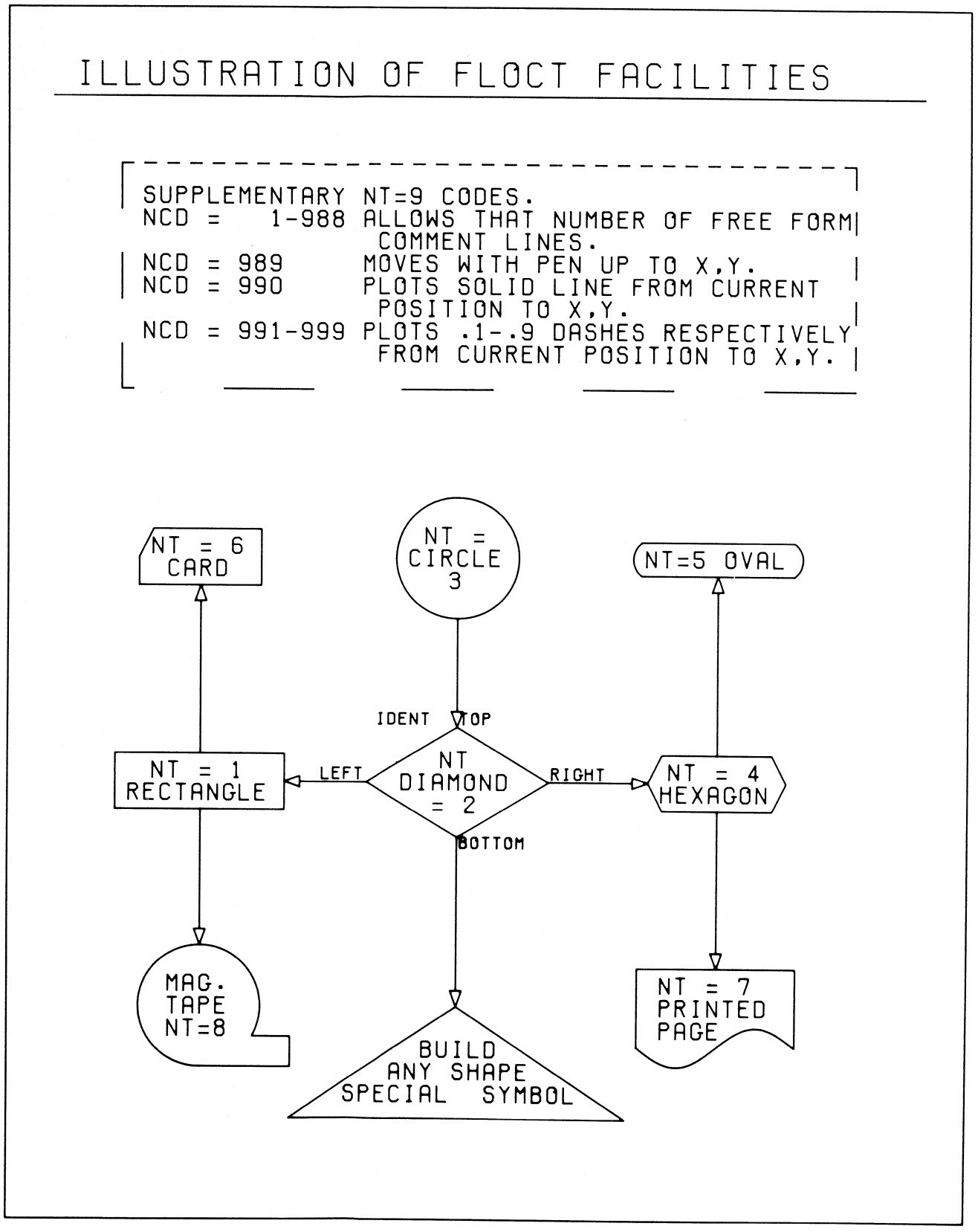


Figure 1-1. FLOCT Capabilities

SECTION 1
PROGRAM CAPABILITIES

GENERAL

FLOCT is a FORTRAN program which generates flowcharts, block diagrams, and decision tables for use in technical presentations and documents. This user's manual describes the capabilities of the program and gives directions for creating the required chart.

NOTE

This manual applies to both inch and metric measurement. Metric parameter values are shown in parentheses in all illustrations. Metric or inch measurement is dependent upon the specific Host Computer Basic Software being used.

Using CalComp Host Computer Basic Software (HCBS), FLOCT can produce the charts offline or online on any CalComp digital graphic system.

OUTPUT FORMAT

The user can use FLOCT to produce a complex chart. The plot consists of a chart title, a combination of nine possible symbols enclosing the text, and arrows connecting the symbols logically. Figures 1-1 and 1-2 are samples of the program capabilities.

AVAILABLE SYMBOLS

Table 1-1 shows the symbol types that can be produced and the corresponding codes which must be input.

The special point symbol is useful for the following purposes:

- Indicates the start or end of an arrow.
- Indicates the start or end of a connecting line.
- Indicates the start of a line of comment.

SYMBOL COMPONENTS

Each symbol on the flowchart has the following components:

- The geometric figure, sized to accommodate the text lines.
- Optional arrows, which may be drawn horizontally or vertically from any of the symbol's sides to another symbol.
- An optional word (or operator) at the start of the arrow, e.g., TRUE, NO, +, 1.
- An optional label over the symbol.
  EXAMPLE: the corresponding statement number of a computer program.
- Lines of text within the figure. There can be up to 999 lines and up to 66 characters on each line.
- An optional 2-character comment inside the top of a diamond symbol.
  EXAMPLE: IF

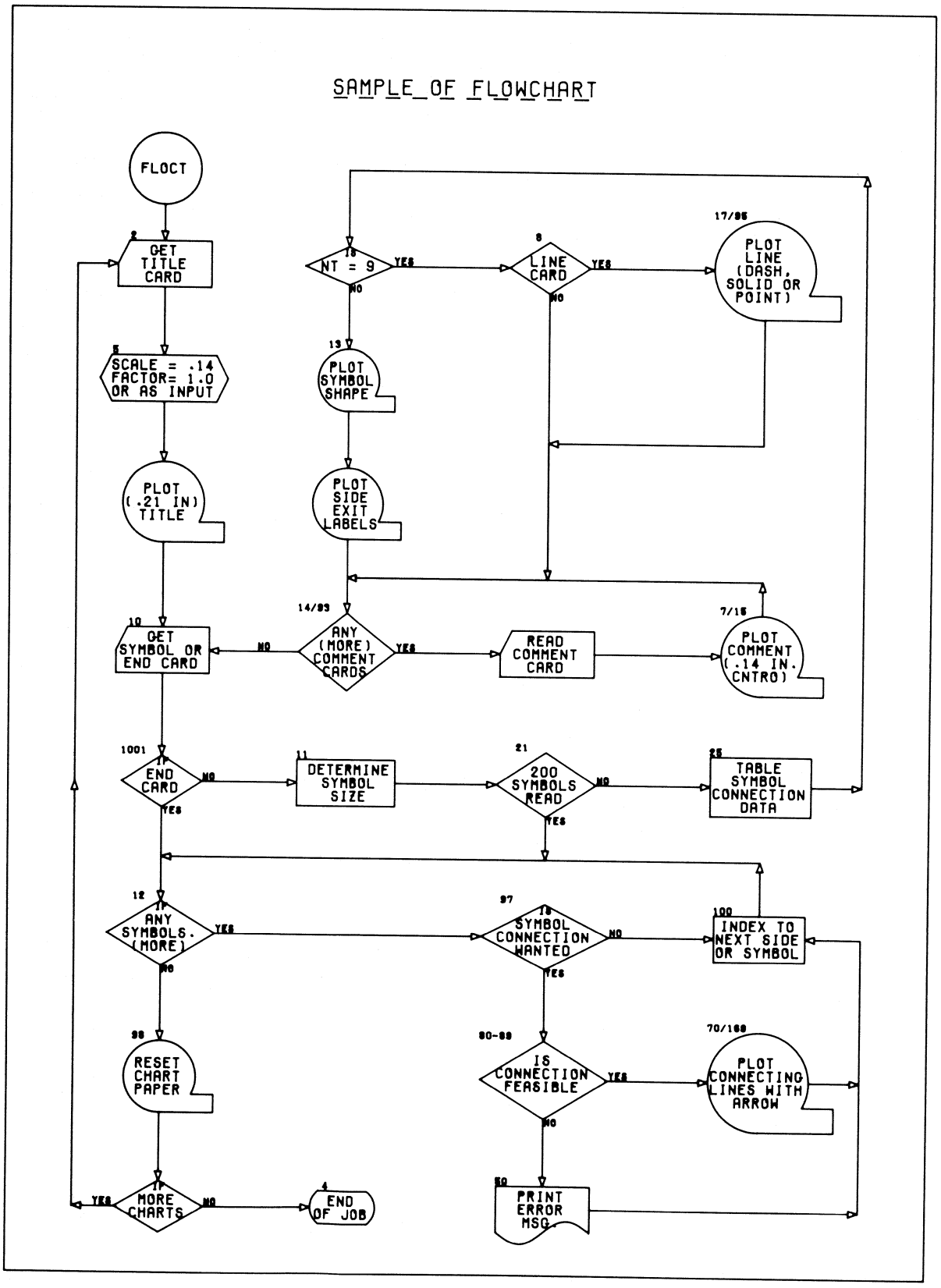


Figure 1-2. Sample Flowchart

Table 1-1. Available Symbols

Input Code	Symbol Name
1	Rectangle
2	Diamond
3	Circle
4	Hexagon
5	Oval
6	Punched Card
7	Printed Page
8	Magnetic Tape
9	Special Point

TABLES AND CHARTS

A table can be generated using text within rectangular symbols. Charts of a user-defined format can also be made, using the following capabilities:

- Lines of comments not within a flowchart symbol.
- Pen carriage movements to a specified page coordinate, with the pen up.
- Solid lines from the current pen position to a specified page coordinate.
- Dashed lines from the current pen position to a specified page coordinate.

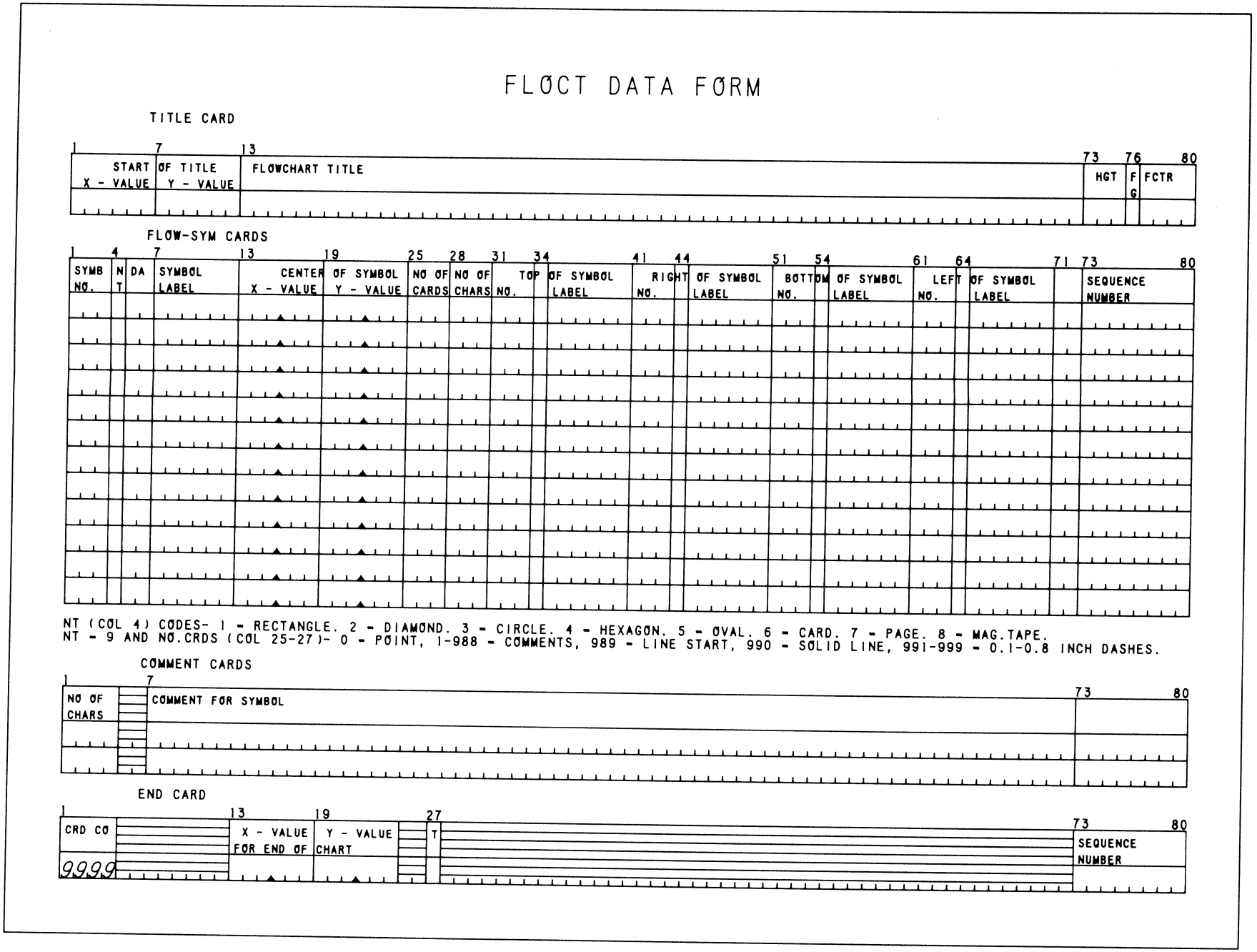


Figure 2-1. Input Form

SECTION 2
INPUT DATA FORMAT

GENERAL

Figure 2-1 is a convenient form showing the formats detailed in Section 2.

CARD TYPES

Four types of data cards are used in the FLOCT program.

TITLE Card	–	generates a title at specified coordinates on the flowchart.
FLOW-SYM Card	–	specifies the symbol to be plotted, its position on the page and the other symbol(s) to which it is connected.
COMMENT Card	–	contains the text placed within the selected symbol or started at a specified point.
END Card	–	terminates input for a chart and provides for generation of additional charts or for termination of the run.

TITLE CARD

This card causes a one-line title to be plotted starting at flowchart page coordinates (XPAGE, YPAGE). The length of the title equals the number of characters in the title times the character height of the title. Only one title line is allowed per page. The TITLE card must be the first card of input for each chart. Each page of a multipage flowchart is considered a separate complete chart.

Card Columns	Data Format	Field Name	Description
1-6	F6.3	XPAGE	X-coordinate, in inches (centimeters)
7-12	F6.3	YPAGE	Y-coordinate, in inches (centimeters)
13-72	A	IBCD	Flowchart page title. Its length cannot exceed 60 characters.

TITLE CARD (cont)

Card Columns	Data Format	Field Name	Description
73-75	F3.3	HGT	If HGT is blank or zero, the character height of the title will be 0.21 inch, and the character height within the geometric symbols will be 0.14 inch. If HGT is nonzero, it is the character height within the symbols. The program scales the title, symbol, and arrowheads, using the ratio: $\frac{HGT}{0.14}$. The distance between the symbols is not affected.
76	I1	FLAG	If this column is zero or blank, each data card will be printed out on the user's line printer. If the column is nonzero, printout of the input data cards is disabled.
77-80	F4.3	FCTR	If FCTR is zero or blank, the flowchart scale will be 1.0. If FCTR is nonzero, the entire flowchart, including the distances between symbols, will be scaled by the factor specified as FCTR.

Card columns 73-80 contain parameters. Deck sequencing cannot be on this card.

FLOW-SYM CARD

The purpose of the FLOW-SYM card is to generate one of the nine standard flowchart symbols at a specified position on a page.

There must be one FLOW-SYM card for each symbol on the flowchart.

The maximum number of FLOW-SYM cards allowable for any flowchart is 200.

Card Columns	Data Format	Field Name	Description
1-3	I3	NO	Unique reference number (0 through 998) for the flowchart symbol to be generated. The integer must be right-justified in the field.
4	I1	NT	Input code (1 through 9) which identifies the flowchart symbol. See Table 1-1.

FLOW-SYM CARD (cont)

Card Columns	Data Format	Field Name	Description
5-6	A	DA	Two characters to be printed inside the top of a diamond (usually IS or IF). This field has no effect if NT is not equal to 2.
7-12	A	AL	Six characters to be printed at the top left on the outside of the flowchart symbol. The characters are usually left-justified, but for clarity may be right-justified in the field. The field is printed with blanks and nonblanks as they appear on the input card.
13-18	F6.3	XPAGE	X-coordinate, in inches (centimeters) of the center of the flowchart symbol.
19-24	F6.3	YPAGE	Y-coordinate, in inches (centimeters) of the center of the flowchart symbol.
25-27	I3	NCD	When NT (card column 4) is not equal to 9, this field defines the number of text lines within the symbol. The entry must be right-justified in the field. When NT is equal to 9, this field takes on the following meanings, depending on its numeric value: NCD = 0-988 Plots the number of text lines indicated, with the first line starting at (XPAGE,YPAGE) and succeeding lines starting directly below. NWD must contain the number of characters in the longest line. NCD = 989 Moves the pen carriage to (XPAGE, YPAGE) with pen up. NWD is not used. NCD = 990 Moves carriage to (XPAGE, YPAGE), with pen down, drawing a solid line. NWD is not used. NCD = 991-999 Moves pen to (XPAGE,YPAGE), drawing a dashed line with dashes of length LD*0.1 inch, where LD is the last digit of NCD. NWD is not used. The integer must be right-justified in the field.

FLOW-SYM CARD (cont)

Card Columns	Data Format	Field Name	Description
28-30	I3	NWD	The maximum number of characters to be printed on a line of text within the symbol. The entry must be right-justified in the field.

NOTE

The next four pairs of data values control the arrows drawn from the symbol. The user specifies which of the four possible exits is used and the target symbol to which the arrow is drawn.

31-33	I3	IREF	Reference number of the symbol to which an arrow is drawn from the top of the current symbol. Leaving IREF blank deletes the arrow and its label.
34	(not used)		
35-40	A	LABEL	Six characters plotted at the exit position of the current symbol. This field describes the condition on which the flow of control follows the arrow. The characters are usually left-justified.
41-43	I3	IREF	Same data as for Columns 31-40, for an arrow drawn from the right side of the symbol.
44	(not used)		
45-50	A	LABEL	
51-53	I3	IREF	Same data as for Columns 31-40, for an arrow drawn from the bottom of the symbol.
54	(not used)		
55-60	A	LABEL	
61-63	I3	IREF	Same data as for Columns 31-40, for an arrow drawn from the left side of the symbol.
64	(not used)		
65-70	A	LABEL	
71-72	(not used)		
73-80	(not used)	none	Available for sequence numbers.

COMMENT CARD

COMMENT cards follow each FLOW-SYM card only when the input code (NT) is 1 through 8, for all NCD values, and when the input code is 9, for NCD values 0 through 988. The number of COMMENT cards must equal NCD (card columns 25-27 of the FLOW-SYM card).

COMMENT CARD

Card Columns	Data Format	Field Name	Description
1-4	I4	NC	Number of characters to be plotted from the IBCD field. NC cannot exceed the value of NWD in the FLOW-SYM card. If the number of characters is less than the value of NWD in the FLOW-SYM card, the text will be centered on the line within the symbol. To left-justify a text line, make NC equal to the value of NWD in the FLOW-SYM card, regardless of the actual number of nonblank characters. The integer must be right-justified in the field.
5-6	(not used)		
7-72	A	IBCD	Up to 66 characters of text.
73-80	Any	none	Available for sequence numbers.

END CARD

Multiple flowcharts may be plotted using one set of data cards. Each flowchart must start with a TITLE card and terminate with an END card.

An END card may indicate that there are no more flowcharts required.

When there are more charts to follow, the END card causes a new origin to be established. The pen is moved to the point (XPAGE,YPAGE), which is considered (0.0,0.0) for subsequent coordinate data. If plotting is done offline, the next sequential Search Address is written on the Plot Tape.

Card Columns	Data Format	Field Name	Description
1-4	I4	IEND	A constant '9999', indicating end of input for the current flowchart.
5-12	(not used)		
13-18	F6.3	XPAGE	X-coordinate of new origin.
19-24	F6.3	YPAGE	Y-coordinate of new origin.
25-26	(not used)		
27	I1	T	If T is zero or blank, the job is terminated (for offline plotting, a 9999 Search Address on the Plot Tape). If T is nonzero, input data is read for the next chart.
28-80	(not used)		

FLOCT TITLE CARD

CARD COLUMNS	DATA FORMAT	FIELD NAME	DESCRIPTION
1-6	F6.3	XPAGE	X-COORDINATE
7-12	F6.3	YPAGE	Y-COORDINATE
13-72	A	IBCD	PAGE TITLE
73-75	F3.3	HGT	CHARACTER HEIGHT
76	I1	FLAG	PRINT INPUT
77-80	F4.3	FCTR	SCALE FACTOR

Figure 3-1. Sample of a Table

SECTION 3
PLANNING THE FLOWCHART

PRELIMINARY LAYOUT

For optimum results from the FLOCT program, a rough layout of the desired flowchart should be drawn on grid paper (10 squares per inch is convenient). The logical flow and the coordinates of the symbol centers can then be determined, and a reference number can be assigned to each symbol.

For every symbol to be generated on a given flowchart, the user must specify the symbol code, the page coordinates of the symbol center, and a reference number. When requesting that a line be drawn to a certain symbol, the user must designate that symbol by its reference number.

Arrows connecting symbols are drawn with horizontal and vertical lines only; they can change direction only once. To connect two symbols when the line between them must change direction more than once, the user may specify Symbol Type 9 for every second point at which a change in direction is required.

A user-determined number of text lines can be plotted within each symbol. The number of characters in any given comment line is variable.

CONTROLLING THE SIZE

The overall size of a flowchart can be increased or decreased by using the scale factor FCTR. The character height in the text can also be controlled by the input variable HGT. This height affects the space required by the title, symbols, and arrowheads.

SYMBOL DIMENSIONS

Table 3-1 shows the X-dimension (width) and Y-dimension (height) for each available symbol. The dimensions assume the default scale factor of 1.0 and the standard character height of 0.14. Other sizes used necessitate scaling the table entries accordingly.

Table 3-1. Symbol Dimensions

Input Code	Symbol Name	Height		Width	
		for one text line	add, for each additional line	add to longest line	minimum
1	Rectangle	0.3	0.2	0.1	1.0
2	Diamond	0.6	0.2	0.4	1.2
3	Circle	0.3	0.2	0.1	0.3
4	Hexagon	0.3	0.2	0.1	1.0
5	Oval	0.3	0.2	0.1	1.0
6	Punched Card	0.3	0.2	0.1	1.0
7	Printed Page	0.3	0.2	0.1	1.0
8	Magnetic Tape	0.3	0.2	0.1	0.3

ARRANGING THE TEXT WITHIN THE SYMBOL

- COMMENT cards with a blank IBCD field can be used to give a symbol more height and space between text lines.

- FLOCT sizes the symbol to accommodate the maximum number of characters, NWD, of a given height, HGT. If more space is desired at the ends of the text lines, or a wider symbol is needed, increase the value of NWD without changing the actual maximum number of characters.

- When the width varies within the symbol (e.g., the diamond), the text lettering may violate some portions of the boundary. Add two or more to the value of NWD to widen the symbol and eliminate the problem.

- When the flowchart must fit on an 8½ by 11-inch page, set the character height HGT at 0.105, which is 3/4 the default size. The title, symbols, and arrowheads will then be at 3/4 size. This is an efficient, readable scale for the page size.

- If symbols overlap, decrease the value of HGT, thereby changing the line length and symbol width. The symbol width is automatically reduced.

GENERATING A TABLE

The following is a sample method for generating a table, using rectangle symbols with COMMENT cards corresponding to the lines of the table:

Step 1. Make a sketch of the table with its entries.

Step 2. Each column, containing all the rows, can be considered a rectangle. Each COMMENT line can be a row of the table. Blank lines and horizontal characters (= or -) form horizontal dividers.

Step 3. Make an accurate sketch on grid paper using the height and width of each rectangle calculated from Table 3-1. Define the origin (0.0, 0.0) at the lower left corner.

Step 4. Calculate the center of each rectangle and the title position based on the sketch generated in Step 3.

Figure 3-1 is an example of a table generated by FLOCT.

OTHER TYPES OF CHARTS

User-defined formats can be developed with the following usage of input data:

- Lines of comments not within a flowchart symbol are made by setting NT = 9 on the FLOW-SYM card, followed by COMMENT cards.
- Pen carriage movements to a specified page coordinate, with the pen up, by setting NCD = 989 on the FLOW-SYM card.
- Solid lines from the current pen position to a specified page coordinate, by setting NCD = 990.
- Dashed lines from the current pen position to a specified page coordinate by setting NCD in the range 991-999.

SECTION 4
ERROR MESSAGES

GENERAL

Table 4-1 is a list of error messages and their meanings.

Table 4-1. Error Messages and their Meanings

Message	Meaning
CANNOT CONNECT BLOCK I TO BLOCK J	The arrow connecting Symbol I to Symbol J requires more than one change in direction. This condition can be corrected by specifying Symbol Type 9 for every second point at which a change in direction is required.
THERE ARE MORE THAN 200 BLOCKS IN THIS PLOT	The input contains more than 200 FLOW-SYM cards. To correct this condition, break the flowchart into two or more parts separated by END and TITLE cards.
THERE IS NO BLOCK _____ TO BE CONNECTED FROM BLOCK _____	Either: A FLOW-SYM card is missing or misidentified; or a reference number (IREF) on a FLOW-SYM card is incorrect.
(Data-error indication, or abnormal run termination)	One of the following has occurred: – The number of COMMENT cards following a FLOW-SYM card does not match the value of NCD. – A TITLE card or END card is missing or is out of sequence. – A numeric field contains alphabetic information.

```
      program curve
      parameter         (ibuflen=1002)
      reaal             x(ibuflen),y(ibuflen),xs(4),ys(4),xp,yp
      integer           i,j,k
      logical           test
      character         kooptext(80),xtext(80),ytext(80),ramdisk(80)
      integer           kooptextlengte,xtextlengte,ytextlengte
      integer           curven,punten
      character*9       filenaamb
      character*80      filenaam
      integer           ramdisklengte
      filenaamb='curve.txt'
      test=.false.
      inquire(file='RAMDISK.TXT',exist=test)
      if(.not. test)stop
      open(11,file='RAMDISK.TXT')
      read(11,10)(ramdisk(i),i=1,80)
10    format(80a1)
      read(11,*)ramdisklengte
      close(unit=11)
      do 1,i=1,Ramdisklengte
1       filenaam(i:i)=Ramdisk(i)
      do 2,i=1,9
2       filenaam(RamDiskLengte+i:RamDiskLengte+i)=filenaamb(i:i)
      test=.false.
      inquire(file=filenaam,exist=test)
      if(.not. test)stop
      open(11,file=filenaam)
c                 koop doorlezen
      do 5,j=1,8
5       read(11,10)(kooptext(i),i=1,80)
      read(11,*)curven
c                 aantal lezen
      read(11,*)punten
      read(11,*)xs(1),ys(1)
      xs(2)=xs(1)
      ys(2)=ys(1)
      do 11,j=2,punten
        read(11,*)xp,yp
        xs(1)=min(xs(1),xp)
        xs(2)=max(xs(2),xp)
        ys(1)=min(ys(1),yp)
11      ys(2)=max(ys(2),yp)
      do 20,j=2,curven
        read(11,*)punten
        do 20,i=1,punten
          read(11,*)xp,yp
          xs(1)=min(xs(1),xp)
          xs(2)=max(xs(2),xp)
          ys(1)=min(ys(1),yp)
20        ys(2)=max(ys(2),yp)
      if(ys(1) .lt. 0)then
```

```
          vs(1)=vs(1)*1.05
         else
          vs(1)=vs(1)*0.95
          endif
        if(vs(2) .gt. 0)then
          vs(2)=vs(2)*1.05
         else
          vs(2)=vs(2)*0.95
          endif
c               ga terug
        rewind(11)
        read(11,10)(kontext(i),i=1,80)
        read(11,*)kooptextlengte
        read(11,10)(xtext(i),i=1,80)
        read(11,*)xtextlengte
        read(11,*)ixastype
        read(11,10)(ytext(i),i=1,80)
        read(11,*)ytextlengte
        read(11,*)iyastype
c
c...plotten
c
        xlen=20.0
        ylen=10.0
        call plots(IDEV,-1,-99)
        call viewport(0.,0.,1.0,0.0,0.75,0)
        call frame
        call symbol(1.0,0.0,-1.0,' ',0.,3)
        call window(-2.0,xlen,-1.5,ylen)
        call gridpunt(0.0,xlen,0.0,ylen,1.)
        if(ixastype .eq. 1)then
          call scale(xs,xlen,2,1)
          call axis(0.,0.,xtext,-xtextlengte,xlen, 0.,xs(3),xs(4))
        else if(ixastype .eq. 2)then
          call scalg(xs,xlen,2,1)
          call lgaxs(0.,0.,xtext,-xtextlengte,xlen, 0.,xs(3),xs(4))
          endif
        if(iyastype .eq. 1)then
          call scale(ys,ylen,2,1)
          call axis(0.,0.,ytext, ytextlengte,ylen,90.,ys(3),ys(4))
        else if(iyastype .eq. 2)then
          call scalg(ys,ylen,2,1)
          call lgaxs(0.,0.,ytext,ytextlengte,ylen,90.,ys(3),ys(4))
          endif

        read(11,*)curven
        call makpen(icolor)
c...file inlezen
        do 60,j=1,curven
          read(11,*)punten
          do 50,i=1,punten
50          read(11,*)x(i),y(i)
```

```
            x(punten+1)=xs(3)
            x(punten+2)=xs(4)
            y(punten+1)=ys(3)
            y(punten+2)=ys(4)
            if(j .eq. 1)then
              lintype=0
            else
              lintype=punten/10
              endif
            if(ixastype .eq. 1 .and. iyastype .eq. 1)then
              call line(x,y,punten,1,lintype,i-1)
            else if(ixastype .eq. 1 .and. iyastype .eq. 2)then
              call lglin(x,y,punten,1,lintype,i-1,1)
            else if(ixastype .eq. 2 .and. iyastype .eq. 1)then
              call lglin(x,y,punten,1,lintype,1,i-1)
            else if(ixastype .eq. 2 .and. iyastype .eq. 2)then
              call lglin(x,y,punten,1,lintype,i-1,0)
              endif
            icolnr=icolor-1
            if(icolor .le. 0)call maxnen(icolor)
            call newnen(icolor)
60          continue
          close(unit=11)
C
C                Kas
          call maxnen(icolor)
          xs(1)=xs(3)
          xs(2)=xs(3)+xlen*xs(4)
          ys(1)=0.0
          ys(2)=0.0
          call plot((xs(1)-xs(3))/xs(4),(ys(1)-ys(3))/ys(4),3)
          call plot((xs(2)-xs(3))/xs(4),(ys(2)-ys(3))/ys(4),2)
          call viewport(0.,0.,1.0,.95,1.0,0)
          call window(0.,20.,-0.2,1.2)
          call symbol(0.,0.,.1,kontext,0.,kopteksteinde)
C
C...einde plot
          call plot(0.,0.,999)
          return
          end
C-----------------------------------------------------------------------
C-----------------------------------------------------------------------
          subroutine gridpunt(xmin,xmax,ymin,ymax,raster)
          real    xmin,xmax,ymin,ymax,raster,x,y
          x=xmin
          y=ymin
10        call plot(x,y,3)
          call plot(x,y,2)
          x=x+raster
          if(x .le. xmax)goto 10
          x=xmin
          y=y+raster
```

```
      if(v .le. vmax)goto 10
      return
      end
C-----------------------------------------------------------------------
      subroutine scale(array,axlen,nots,inc)
      real    array(*),low,high,axlen
      integer nots,inc
      common          /border/low,high
C
      low=array(1)
      high=array(1)
      i=1
10    i=i+inc
      if(i .gt. nots)goto 20
      low=amin1(low,array(i))
      high=amax1(high,array(i))
      goto 10
20    array(nots%inc+1)=low
      array((nots+1)%inc+1)=(high-low)/axlen
      return
      end
```

De datalijst heeft u nu niet meer nodig.
Probeer ook:

```
50 Y(Q)=-60-50*(SIN(((Q/2.5)-.7)/Q-.9)) : X(Q)=Q*5+30
```

voor een knikkerputje.
Heel mooi is de volgende formule:

```
70 Y(Q)=-110-.5*(Q*Q)*(SIN(Q)/Q-.9) : X(Q)=7*Q+10
```

Een kubus

Het programma hieronder laat zien hoe een object rond alle drie de assen kan worden gedraaid. Dit programma is opvallend door de snelheid. Dit ondanks de vele berekeningen die voor elk punt opnieuw moeten worden gemaakt. Bij een draaiing om drie assen worden de berekeningen die gemaakt moeten worden gemaakt, om het projectiepunt van elk punt te bepalen, een stuk langer. In dit programma wordt altijd eerst gedraaid om de X-as, daarna om de Y-as en tot slot rond de Z-as. Deze volgorde kan worden veranderd, maar is bepalend voor voor het uiteindelijke resultaat. Door echter de juiste draaiingshoeken te kiezen kan de gebruiker elke stand van het object verkrijgen.
De projectieformule is op de volgende wijze gevonden:
De matrix van een projectie op het X-Y vlak (Z=0) na een rotatie om de Z-as, na een rotatie om de Y-as, na een rotatie om de X-as is berekend. Deze matrix is gevonden door eerst de matrices voor rotatie om de Y-as en de X-as te vermenigvuldigen. Vervolgens wordt het resultaat hiervan vermenigvuldigd met een matrix voor een rotatie om de Z-as (over hoek C). Het resultaat is:

COS(B)*COS(C)	SIN(A)*SIN(B)*COS(C) +COS(A)*SIN(C)	-COS(A)*SIN(B)*COS(C)+ SIN(A)*SIN(C)
-COS(B)*SIN(C)	-SIN(A)*SIN(B)*SIN(C)+ COS(A)*COS(C)	COS(A)*SIN(B)*SIN(C)+ SIN(A)*COS(B)
SIN(B)	-SIN(A)*COS(B)	COS(A)*COS(B)

Deze matrix wordt vermenigvuldigd met de projectiematrix (Z=0).

De gebruiker kan in het volgende programma kiezen uit twee mogelijkheden: datalijst of data-invoer. Kiest u voor de datalijst dan hoeft u verder niets meer te doen. Het programma heeft een stel gegevens klaar om een kubus te tekenen waarmee getoond kan worden wat het programma allemaal kan. Wilt u een ander voorwerp, dan kan dat door de gegevens in te voeren na het kiezen van data-invoer. De gegevens die u dan invoert blijven echter niet in het programma staan. Als u veel verschillende voorwerpen met veel gegevens wilt gebruiken bij dit of een van de volgende programma's dan kunt u deze gegevens het beste opslaan in aparte bestanden. Lees hiervoor het hoofdstuk over het werken met bestanden.

```
10 GRAPHICS 0
20 PRINT "DATALIJST (0) OF DATAINVOER (1)";
25 INPUT E
30 IF E=1 THEN 140
40 RESTORE
50 READ L
60 DIM X(L),Y(L),Z(L),T(L)
70 FOR H=1 TO L
80 READ A,B,C,D
90 X(H)=A:Y(H)=B
100 Z(H)=C:T(H)=D
120 NEXT H
130 GOTO 250
140 PRINT "AANTAL PUNTEN";:INPUT L
150 DIM X(L),Y(L),Z(L),T(L)
160 FOR H=1 TO L
170 PRINT "X-COORD. PUNT ";H;" ";:INPUT A
180 X(H)=A
190 PRINT "Y-COORD. PUNT ";H;" ";:INPUT B
200 Y(H)=B
210 PRINT "Z-COORD. PUNT ";H;" ";:INPUT C
220 Z(H)=C
230 PRINT "LIJN (1) OF VERPLAATS (0)";
235 INPUT D:T(H)=D
240 NEXT H
250 GRAPHICS 8+32
260 PRINT "   INVOER IN GRADEN"
270 PRINT "ROTATIE OM X-AS ";:INPUT A1
280 PRINT "ROTATIE OM Y-AS ";:INPUT B1
290 PRINT "ROTATIE OM Z-AS ";:INPUT C1
300 GRAPHICS 8+16
310 GOSUB 540:REM HOEKBEREKENING
320 GOSUB 420:REM TEKENEN
330 FOR WA=1 TO 3000:NEXT WA
340 GOTO 250
350 END
360 DATA 21
370 DATA 0,0,0,0,0,50,0,1,50,50,0,1,50
375 DATA 0,0,1,50,0,50,1,50,50,50,1
380 DATA 0,50,50,1,0,0,50,1,50,0,50,1
385 DATA 50,50,50,0,50,50,0,1,0,50,0,0
390 DATA 0,50,50,1,0,0,50,0,0,0,0,1,50
```

```
400 DATA 2,2,0,1,48,2,0,1
410 RE TEKENEN
420 PLOT 125,125
430 FOR H=1 TO L
440 GOSUB 500:REM PROJECTIE
450 IF T(H)=0 THEN PLOT PX,PY
460 IF T(H)=1 THEN DRAWTO PX,PY
470 NEXT H
480 RETURN
490 REM PROJECTIE
500 PX=(X(H)*F+Y(H)*G+Z(H)*E)+125
510 PY=(X(H)*I+Y(H)*J+Z(H)*K)+75
520 RETURN
530 REM HOEKBEREKENING
540 A=(A1*3.14)/180
545 B=(B1*3.14)/180:C=(C1*3.14)/180
550 F=COS(B)*COS(C)
560 G=SIN(A)*SIN(B)*COS(C)+COS(A)*SIN(C)
570 E=SIN(A)*SIN(C)-COS(A)*SIN(B)*COS(C)
580 I=COS(B)*SIN(C)*-1
590 J=COS(A)*COS(C)-SIN(A)*SIN(B)*SIN(C)
600 K=COS(A)*SIN(B)*SIN(C)+SIN(A)*COS(C)
610 RETURN
```

De eerste regels zorgen voor een leeg scherm en vragen de gebruiker of hij zelf gegevens wil invoeren of gebruik wil maken van de al in het programma aanwezige kubus. Als er gebruik wordt gemaakt van de kubus springt de computer na het dimensioneren en inlezen van de arrays over naar regel 250 (regel 130).
De regels 140-240 zorgen ervoor dat de gebruiker zelf een vorm kan opgeven waarmee de computer vervolgens aan het werk kan gaan.
De regels 260-340 zorgen in een lus dat de gebruiker de draaihoeken op kan geven en zorgen door middel van twee subroutines voor het rekenen en voor het tekenen.
De regels 360-400 vormen de datalijst. Het gegeven achter de dataopdracht van regel 360 geeft alleen het aantal te lezen gegevens in de datalijst aan.
De regels 410-480 vormen de tekensubroutine. De grafische cursor wordt naar de juiste plaats gebracht (regel 420) en in een lus worden de gegevens een voor een afgewerkt. Eerst wordt het rekenwerk gedaan. Dit gebeurt in een aparte subroutine die op regel 490 begint. Aan deze rekenregels (500 en 510) kunt u zien dat er al wat rekenwerk is gedaan. Als er niet van te voren al wat uitgerekend was, had de computer voor elk punt moeten berekenen:

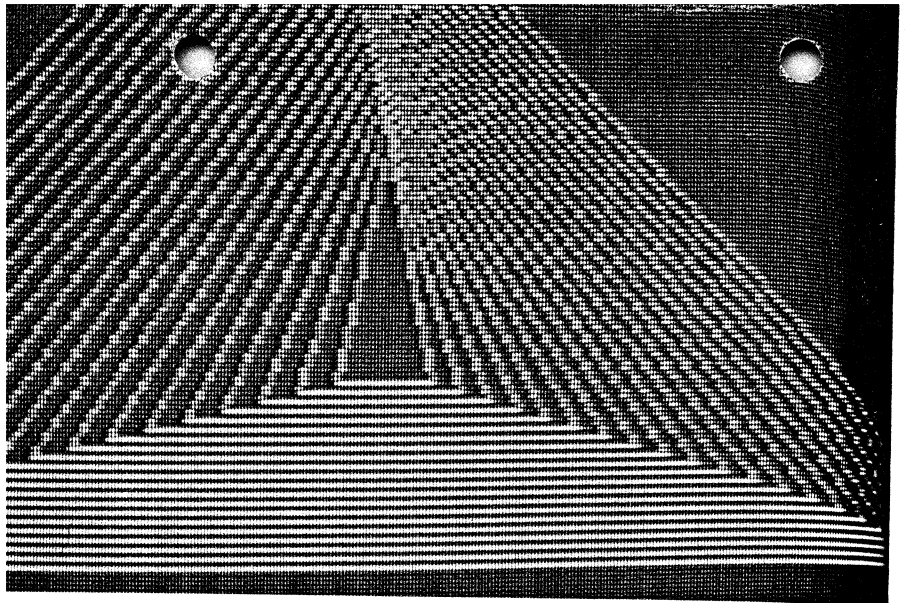
```
PX = X(H)*COS(B)*COS(C)+Y(H)*(SIN(A)*SIN(B)*COS(C)+COS(A)
     *SIN(C)) +Z(H)*(SIN(A)*SIN(C)-COS(A)*SIN(B)*COS(C))
PY = X(H)*-(COS(B)*SIN(C))+Y(H)*(COS(A)*COS(C)-SIN(A)*SIN(B)*
     SIN(C))+Z(H)*(COS(A)*SIN(B)*SIN(A)+SIN(A)*COS(C))
```

De berekeningen hiervoor staan grotendeels in de subroutine die buiten de tekenlus wordt aangeroepen, en de regels 530-610 beslaat.
Na de projectiesubroutine wordt gekeken of de laatste van de vier gegevens een 0 of een 1 is. Is het een 0 dan wordt er geen lijn getrokken, is het een 1, dan wordt er wel een lijn getrokken.

Een piramide

De mogelijkheden van het programmeren 3D zijn nog lang niet uitgeput. Neemt u het volgende programma over. Gebruik als basis het vorige programma, want er zijn veel overeenkomsten.

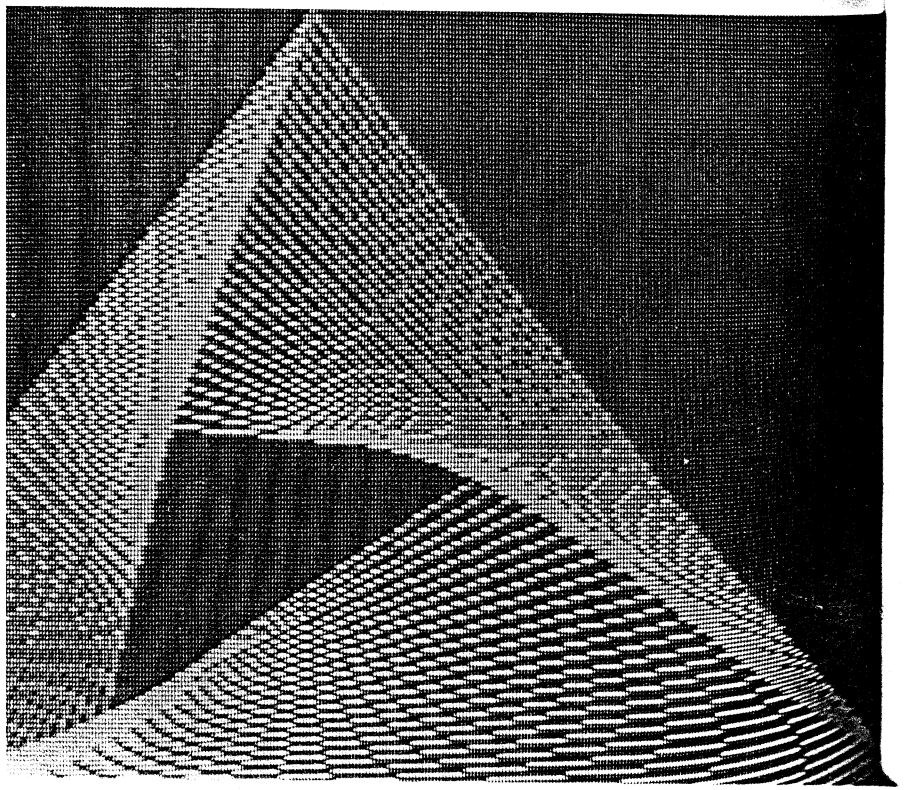
```
10 GRAPHICS 0
20 PRINT "DATALIJST (0) OF DATAINVOER (1)";
25 INPUT E
30 IF E=1 THEN 140
40 RESTORE
50 READ L
60 DIM X(L),Y(L),Z(L),T(L)
70 FOR H=1 TO L
80 READ A,B,C,D
90 X(H)=A:Y(H)=B
100 Z(H)=C:T(H)=D
120 NEXT H
130 GOTO 250
140 PRINT "AANTAL PUNTEN";:INPUT L
150 DIM X(L),Y(L),Z(L),T(L)
160 FOR H=1 TO L
170 ? "X-COORD. PUNT ";H;" ";:INPUT A
180 X(H)=A
190 ? "Y-COORD. PUNT ";H;" ";:INPUT B
200 Y(H)=B
210 ? "Z-COORD. PUNT ";H;" ";:INPUT C
220 Z(H)=C
230 PRINT "LIJN (1) OF VERPLAATS (0)";
235 INPUT D:T(H)=D
240 NEXT H
250 GRAPHICS 8+32
260 PRINT "   INVOER IN GRADEN"
265 PRINT "ROTATIE OM X-AS ";:INPUT A1
270 PRINT "ROTATIE OM Y-AS ";:INPUT B1
275 PRINT "ROTATIE OM Z-AS ";:INPUT C1
280 ? "SCHAAL X-RICHTING";:INPUT SX
285 ? "SCHAAL Y-RICHTING";:INPUT SY
290 ? "SCHAAL Z-RICHTING";:INPUT SZ
```

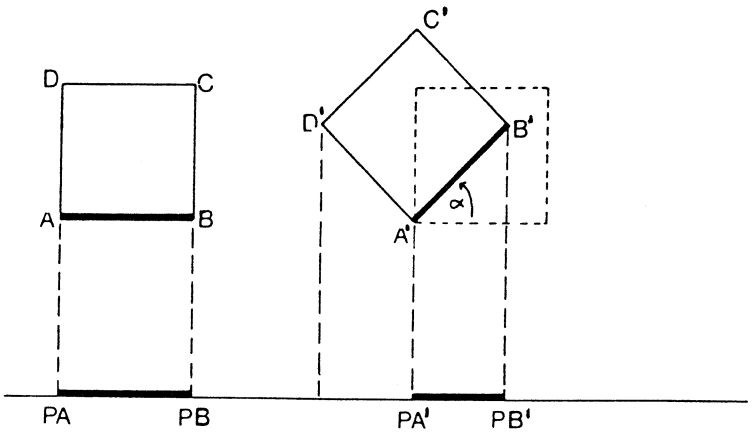


Tunnel programma *blz. 251*

ogramma *blz. 251*

fotografie: Jorn van Engelen

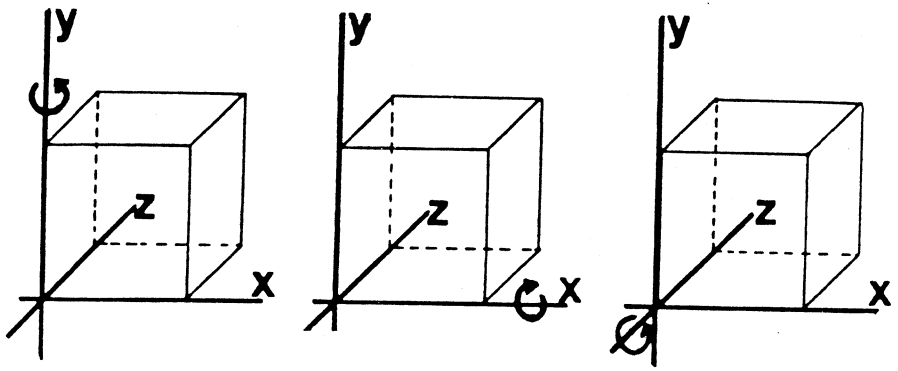




rotatie en projectie van een vierkant

De punten A, B, C en D zijn de hoekpunten van het vierkant zonder dat het is gedraaid. De punten A', B', C' en D' zijn de hoekpunten van het vierkant na draaiing. De draaiing vond plaats over een hoek van 45 graden. De afstand AB is 1. Het figuur is een vierkant dus zijn alle andere zijden ook precies 1 eenheid lang.
Bij een projectie zonder draaiing is de afstand PA-PB (de afstand tussen de projectiepunten van A en B) gelijk aan 1. De afstand tussen PD en PC is ook 1. De afstand tussen PA en PD en tussen PB en PC is 0.
Na het draaien ziet het er heel anders uit. De afstand PA' en PB' is kleiner dan 1. Hoeveel kleiner dat is, is afhankelijk van de hoek waarover is gedraaid. Bij een draaiing over een hoek van 45 graden is de afstand tussen PA' en PB' ongeveer 0.7. Voor de wiskundige lezers: deze afstand is precies gelijk aan 0.5*SQR(2), oftewel COS(RAD(45)). Hoe meer er gedraaid wordt, hoe kleiner de afstand. Als er meer dan 90 graden wordt gedraaid, wordt de afstand zelfs negatief (PB' ligt dan links van PA').
Op deze wijze zijn de verschillende afstanden op de projectielijn uit te rekenen en zijn de verschillende projectiepunten te berekenen.

verschillende rotaties: om de X-as, om de Y-as en om de Z-as.



verschillende rotatiemogelijkheden

De matrix van een rotatie over een hoek van A graden om de X-as ziet er als volgt uit:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & COS(A) & SIN(A) \\ 0 & -SIN(A) & COS(A) \end{pmatrix}$$

U ziet dat het punt (1,0,0) bij deze rotatie onveranderd blijft. Het punt (0,1,0) wordt veranderd in het punt (0,COS(A),-SIN(A)). Als we voor A bijvoorbeeld 30 graden nemen dan wordt het punt (0,1,0) veranderd in (0,0.866,-0.5) (0.866 is een benadering van 0.5*SQR(3), dat is de cosinus van 30 graden. 0.5 is de sinus van 30 graden).
De matrix van een rotatie over een hoek van B graden om de Y-as ziet er als volgt uit:

$$\begin{pmatrix} COS(B) & 0 & -SIN(B) \\ 0 & 1 & 0 \\ SIN(B) & 0 & COS(B) \end{pmatrix}$$

Bij deze rotatie blijft het punt (0,1,0) onveranderd.
De matrix voor een rotatie over een hoek van C graden om de Z-as is als volgt:

$$\begin{pmatrix} COS(C) & SIN(C)- & 0 \\ -SIN(C) & COS(C) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Hier blijft het punt (0,0,1) onveranderd.

Een kubus

Het volgende programma tekent een kubus die niet recht van voren wordt bekeken. De kubus wordt eerst gedraaid om de X-as en daarna om de Y-as. De matrix die bij deze gecombineerde draaiing hoort, verkrijgt men door de matrix van een rotatie om de X-as te vermenigvuldigen met de matrix van een rotatie om de Y-as.
Een rotatie om de X-as over een hoek van A graden gevolgd door een rotatie om de Y-as over een hoek van B graden geeft:

$$\begin{pmatrix} COS(B) & 0 & -SIN(B) \\ 0 & 1 & 0 \\ SIN(B) & 0 & COS(B) \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & COS(A) & SIN(A) \\ 0 & -SIN(A) & COS(A) \end{pmatrix}$$

Het resultaat van deze vermenigvuldiging moet nog vermenigvuldigd worden met de projectiematrix (de matrix die ervoor zorgt dat de Z-coördinaten gelijk aan 0 worden). Vervolgens moet het resultaat vermenigvuldigd worden met de kolommatrix van het te roteren punt (X,Y,Z) van de kubus.

$$\begin{pmatrix} PX \\ PY \\ PZ \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} COS(B) & SIN(A)*SIN(B) & -COS(A)*SIN(B) \\ 0 & COS(A) & SIN(A) \\ SIN(B) & -SIN(A)*COS(B) & COS(A)*COS(B) \end{pmatrix} * \begin{pmatrix} X \\ Y \\ z \end{pmatrix}$$

Het eindresultaat:

PX = X*COS(B) + Y*SIN(A)*SIN(B) - Z*COS(A)*SIN(B)
PY = Y*COS(A) + Z*SIN(A)
PZ = 0

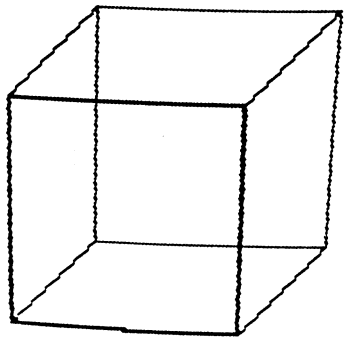
De waarden PX, PY en PZ staan voor de waarden van respectievelijk de X-coordinaat, de Y-coordinaat en de Z-coordinaat van het punt na projectie. Het punt heeft een Z-coordinaat die 0 is en ligt dus op het projectievlak (in ons geval het beeldscherm). Een rotatie die eerst om de Y-as gaat met een hoek van B graden en daarna om de X-as met een hoek van A graden en een projectie geeft als resultaat:

PX = X*COS(B) - Z*SIN(B)
PY = X*SIN(A)*SIN(B) + Y*COS(A) + Z*SIN(A)*COS(B)
PZ = 0

Dit programma laat een kubus in verschillende standen zien.

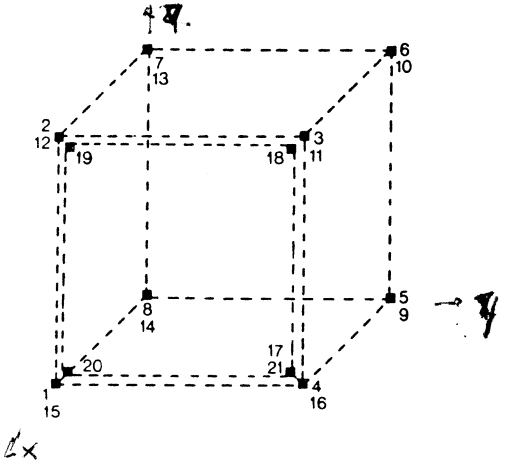
```
10 GRAPHICS 8:COLOR 1
20 DIM X(21),Y(21),Z(21)
30 FOR D=1 TO 21
40 READ A,B,C
50 X(D)=A:Y(D)=B:Z(D)=C
60 NEXT D
```

```
70 PRINT "DRAAIING OM Y-AS";:INPUT A
80 PRINT "DRAAIING OM X-AS";:INPUT B
85 TRAP 250
90 A=(A*3.14)/180
100 B=(B*3.14)/180
110 PLOT 100,130
120 K=COS(A):L=SIN(A)*SIN(B):M=SIN(A)*COS(B):N=COS(B):J=SIN(B)
130 FOR D=1 TO 21
140 PX=X(D)*K+Y(D)*L-Z(D)*M
150 PY=Y(D)*N+Z(D)*J
160 DRAWTO PX+100,130-PY
170 NEXT D
180 FOR WA=1 TO 3000:NEXT WA
190 GRAPHICS 8
200 GOTO 70
210 END
250 PRINT "KUBUS KOMT BUITEN BEELD."
260 PRINT "VUL NIEUWE WAARDEN IN S.V.P."
270 GRAPHICS 8:GOTO 70
300 DATA 80,0,0,80,80,0,0,80,0,0,0,0
305 DATA 1,1,1,79,1,0
310 DATA 79,79,0,1,79,0,1,1
315 DATA 0,80,0,0,80,0,80
320 DATA 80,80,80,80,80,0,80,80,80
330 DATA 0,80,80,0,80,0,0,0,0,0,0,80
340 DATA 80,0,80,0,0,0,80,0,80,80
```



een kubus getekend door bovenstaand programma

In regel 20 wordt voldoende plaats gemaakt voor het tekenen van de kubus. In de regels 30-60 worden de verschillende gegevens voor de kubus in een array geplaatst. Daardoor gaat het tekenen sneller.
De gegevens die in de datalijst zijn opgenomen, zijn de punten waarnaar een lijn al dan niet getrokken moet worden (telkens een X-, Y- en Z-coordinaat). In het voorbeeldprogramma kon volstaan worden met het telkens trekken van een lijn. Een kubus kan worden getekend door 16 lijnen te trekken. Om te zorgen dat de voorste lijnen ietsje dikker worden (hetgeen het 3D effect vergroot) zij. 21 lijnen nodig volgens onderstaand schema:



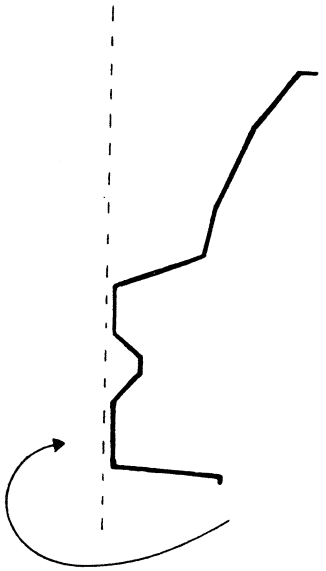
Als u andere figuren dan kubussen wilt tekenen, zult u soms te maken krijgen met tekeningen waarbij het veel eenvoudiger is om 'het potlood' even te verplaatsen zonder een lijn te trekken. U moet dan elk coordinatenstel voorzien van een vierde getal (0 of 1) voor het al dan niet tekenen van de lijn. Het is het handigste om daarbij altijd 0 aan te geven als er geen lijn moet worden getrokken en 'het potlood' alleen moet worden verplaatst . Een 1 staat dan voor het trekken van een lijn. Een eenvoudige IF..THEN..opdracht zorgt ervoor dat op de juiste plaatsen de lijn wordt getrokken.
De regels 70 en 80 vragen de gebruiker om welke hoek de vorm gedraaid moet worden voordat hij geprojecteerd en getekend wordt.
De regels 90 en 100 zijn wat cryptisch. De meest gebruikte wijze om hoeken aan te geven is in graden. Er gaan 360 graden in een volle cirkel. Om een programma een beetje gebruikersvriendelijk te maken is het in elk geval nodig dat de gebruiker de hoeken in een voor hem/haar bekende grootheid op kan geven. In dit geval graden. De Atari werkt echter met radialen. In een volle cirkel gaan 2*PI radialen. PI is ongeveer gelijk aan 3.14. In de regels 90 en 100 rekent de Atari zelf de hoek in graden om in radialen. Dit is de tweede methode om het verschil tussen graden en radialen te overkomen. De andere methode vindt u in de paragraaf over dieptesuggestie door schaduwwerking.
Regel 110 verplaatst de grafische cursor naar de beginpositie.
Regel 120 moet u in samenhang met de regels 140 en 150 lezen. De laatste twee regels worden voor elk punt herhaald. Regel 120 wordt maar een keer per tekening uitgevoerd. Door op deze wijze een deel van de berekeningen naar een programmaregel te verplaatsen, waar zij maar een keer uitgevoerd hoeven te worden in plaats van vele keren, kan het tekenen vele malen sneller gaan. *De berekening die de computer bij elk te tekenen punt moet maken is nu veel korter.* Test u dit uit door in regel 140 en 150 de oorspronkelijke berekening te plaatsen:

```
140 PX=X(D)*COS(A)+Y(D)*SIN(A)*SIN(B)-Z(D)*SIN(A)*COS(B)
150 PY=Y(D)*COS(B)+Z(D)*SIN(B)
```

Ziet u hoeveel langzamer het tekenen gaat? Bij het maken van uw eigen tekenprogramma's waar veel rekenwerk in voorkomt moet u hier goed op letten: laat zo veel mogelijk rekenwerk van te voren (dat wil zeggen buiten de tekenlus) doen. Een herhalingslus moet zo kort mogelijk zijn om snelle tekeningen te kunnen maken.
Regel 160 zorgt voor het uiteindelijke tekenwerk. Regel 160 zou eigenlijk overbodig moeten zijn. Door de coordinaten zorgvuldig te kiezen kan ervoor worden verzorgd dat een verschuiving niet meer nodig is. Anderzijds heeft een verschuivingsfactor het voordeel dat de vorm over het scherm heen en weer kan worden bewogen.
Regel 180 wacht een tijdje voordat regel 190 weer terugschakelt naar scherm 8, zodat gebruiker een nieuw stel hoeken op kan geven.

Een wijnglas

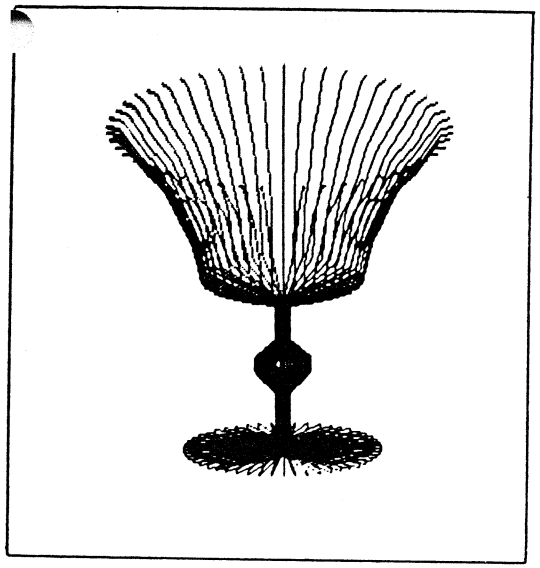
Door het vorige programma wat uit te breiden kunt u het uzelf nog gemakkelijker maken. Het onderstaande programma tekent niet alleen een door u bedacht voorwerp, het bedenkt zelfs hoe het voorwerp er uit moet zien. Het enige dat u hoeft in te voeren is een willekeurige gekromde lijn. De computer draait deze lijn telkens een paar graden rond. De rotatie vindt plaats rond de Y-as.



Na 360 graden is de lijn weer op zijn oude standpunt.

aankijkt, lijkt het of de computer een eenvoudige 2D tekening heeft gemaakt. Maar de computer heeft tijdens het draaien de verkregen gegevens opgeslagen in een array. Als u na het vormen van het figuur de computer vraagt het voorwerp te roteren rond de X-as kunt u het voorwerp van verschillende kanten bekijken.

```
10 GRAPHICS 8+16
20 SETCOLOR 2,0,0:COLOR 1
30 DIM X(9),Y(9),Z1(33,9),Z2(33,9)
40 FOR Q=1 TO 9
50 READ A,B
60 X(Q)=A:Y(Q)=B
70 NEXT Q
80 FOR D=1 TO 32
90 PLOT 125,170
100 S=SIN(D*0.2)
110 C=COS(D*0.2)
120 FOR A=1 TO 9
130 Z1(D,A)=S*X(A)
140 Z2(D,A)=C*X(A)
150 DRAWTO Z1(D,A)+125,Y(A)+160
160 NEXT A
170 PLOT 125,160
180 NEXT D
190 FOR B=0.5 TO 0.8 STEP 0.3
200 GRAPHICS 8+16:SETCOLOR 2,0,0
210 PLOT 125,160
220 N=COS(B):J=SIN(B)
230 FOR D=1 TO 32
240 PLOT 125,160
250 FOR A=1 TO 9
260 PX=Z1(D,A)
270 PY=Y(A)*N+Z2(D,A)*J
280 DRAWTO PX+125,PY+160
290 NEXT A:NEXT D
295 FOR WA=1 TO 3000:NEXT WA
300 NEXT B
310 GOTO 310
500 DATA 40,0,4,6,4,-30,10,-45,4,-55,4
510 DATA -70,30,-80,35,-106,55,-150
```



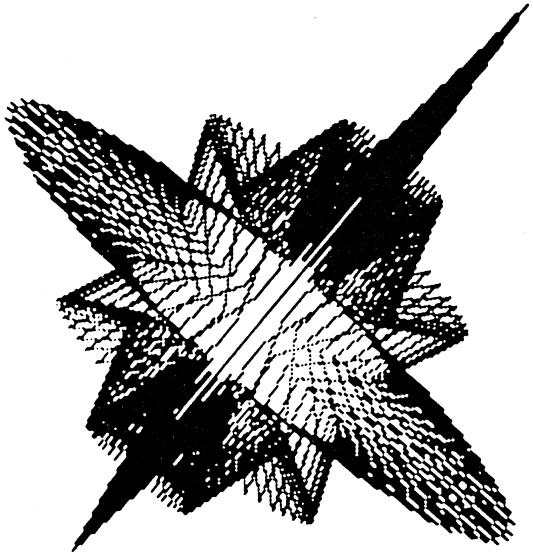
t programma laat als voorbeeld een wijnglas zien. Nadat het glas eerst in 2D
getekend, waarbij de gegevens voor de 3D weergave gelijk worden berekend,
rdt vervolgens het glas in verschillende 3D standen getoond. De hoeken
arover het glas om de X-as geroteerd wordt, kunt u (in radialen) aflezen in
el 190. Roteren rond de Y-as heeft niet zoveel zin, omdat het voorwerp
kregen is door rotatie rond de Y-as en perfect symetrisch is. Door de
aiing rond de X-as kunt u het glas van binnen bekijken.
gel 30 reserveert ruimte voor de 2D en 3D weergave.
regels 40-70 lezen de gegevens uit de datalijst. De gegevens bestaan uit
ens twee getallen die een punt van de hiervoor getekende lijn aangeven.
regels 80-180 tekenen 32 keer alle negen punten van de lijn. Telkens wordt
ijn een beetje gedraaid. In de 2D weergave is dat te zien doordat de lijn ver-
uift.
regels 130 en 140 plaatsen de 3D gegevens in de array.
gel 150 zorgt voor het tekenen van de 2D uitvoering.
gel 170 zorgt voor het verplaatsen van de grafische cursor.
naf regel 190 wordt de 3D weergave verzorgd. Heeft u liever dat u zelf de
k waarover het wijnglas draait kunt opgeven, dan kunt u regel 190, 200 en
veranderen zodat er een INPUT opdracht komt te staan (daarvoor moet u
elijk naar een schermmodus 0 overschakelen, al dan niet als tekstvenster).
regels 210-290 zijn vergelijkbaar met de regels 80-180. Nu hoeft er echter
s uitgerekend te worden, maar kan de computer direkt de gegevens uit de
y gebruiken om te gaan tekenen. In regel 220 wordt vast een deel van het
nwerk gedaan. Dit is eens stuk eenvoudiger dan in het kubusprogramma.
komt doordat er slechts over een as gedraaid hoeft te worden.

in plaats van een wijnglas een ander figuur te krijgen hoeft u alleen de ge-
ns te veranderen. Let u hierbij wel op: als u meer gegevens plaatst, moet

120 en 250). Teve moet de dimensionering van de arrays eventue worden aangepast (regel 30).
Verander in het vorige programma de volgende regels:

```
 30 DIM X(13),Y(13),Z1(33,13),Z2(33,13)
 40 FOR Q=1 TO 13 -120 FOR A=1 TO 13
250 FOR A=1 TO 13
500 DATA    0,10,7,-30,15,0,22,-30,40,-10,    30,-40,75,-50,30,-60,40,-
    90,22,-70,15, -100,7,-70,0,-150
```

Voor andere voorwerpen kunt u de gegevens zelf veranderen. De gegevens horen twee aan twee bij elkaar. Telkens een X-coordinaat en een Y-coordinaat van een punt van de gekromde lijn die moet worden gedraaid. Let op de ruimtereservering voor de verschillende arrays en het aantal keren dat de lussen moeten worden doorlopen.



U kunt het uzelf nog gemakkelijker maken door zelfs de gekromde lijn niet punt voor punt op te geven, maar de computer te vragen om aan de hand van een door u opgegeven formule de punten te bepalen. Op deze wijze krijgt u mooie ribbelvlakken. Vervang, van het al volgens bovenstaande aanwijzingen veranderde programma, de volgende regels:

```
 50 Y(Q)=-50-70:SIN(Q-9.9)/Q):X(Q)=Q*7+10
210 PLOT 125,130
240 PLOT 125,130
```

plays a series of $n - 1$ line segments that connect the n adjacent coordinates from position $(x[1], y[1])$ to position $(x[n], y[n])$.

We can implement the *polyline* command for $n > 2$ with a procedure that makes repeated calls to the line-drawing algorithm. Each successive call to the line-drawing algorithm passes the coordinate pair needed to plot the next line segment. The line-drawing algorithm is accessed by this procedure a total of $n - 1$ times.

3–5 Fill Areas

When shading or color patterns are to be applied to areas of a scene or graph, it is convenient for a user to be able to specify the area that is to be filled. Although fill patterns could be applied to the interiors of polygon borders defined with a line command, processing is simplified if a separate procedure is used to define a fill area. This approach allows a designated area to be immediately flagged as one that is to be displayed with a specified interior.

We introduce the following command to define a polygon fill area:

```
fill_area (n, x, y)
```

The area to be filled is inside the boundary defined by the series of n connected line segments from $(x[1], y[1])$ to $(x[n], y[n])$ and back to $(x[1], y[1])$.

Implementation of the *fill_area* command in a graphics package depends on the type of fill that is to be used to display the area. A user might want the area left blank or filled with a solid color or some pattern. When the interior of the area is left blank, *fill_area* simply produces the boundary outline. This is analogous to using the *polyline* procedure with the starting and ending coordinates set to the same values.

3–6 Circle-Generating Algorithms

Since the circle is a common component in many types of pictures and graphs, procedures for generating circles (and ellipses) are often included in graphics packages. The basic parameters that define a circle are the center coordinates (xc, yc) and the radius r (Fig. 3–12). We can express the equation of a circle in several forms, using either Cartesian or polar coordinate parameters. Figure 3–13 shows the relationship between Cartesian and polar parameters.

FIGURE 3–12
Circle with center coordinates *(xc, yc)* and radius *r*.

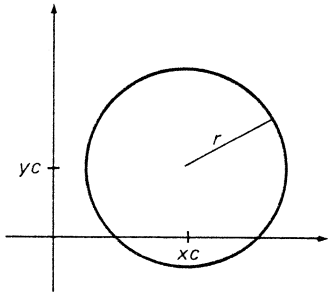
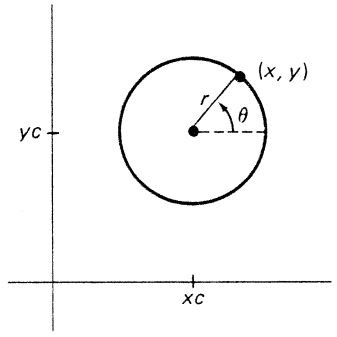


FIGURE 3–13
Relationship between Cartesian and polar coordinates.



Circle Equations

A standard form for the circle equation is the Pythagorean theorem:

$$(x - xc)^2 + (y - yc)^2 = r^2 \tag{3-16}$$

This equation could be used to draw a circle by stepping along the x axis in unit steps from $xc - r$ to $xc + r$ and calculating the corresponding y values at each position as

$$y = yc \pm \sqrt{r^2 - (x - xc)^2} \tag{3-17}$$

Obviously, this approach involves considerable computation at each step, and the

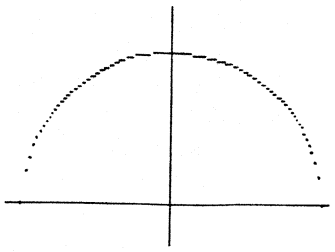


FIGURE 3–14
Positive half of a circle plotted with Eq. 3–17 and *(xc, yc)* = (0, 0).

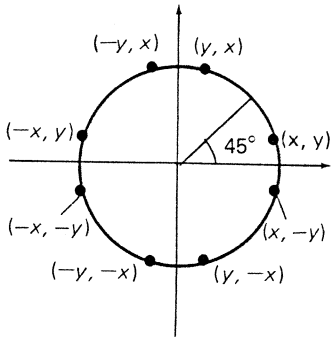


FIGURE 3–15
Symmetry of a circle. Calculation of a point *(x, y)* on the first one-eighth circle segment also yields the seven additional points shown on the circle.

spacing between plotted pixel positions is not uniform, as demonstrated in Fig. 3–14. We could adjust the spacing by interchanging x and y (stepping through y values and calculating x values) whenever the absolute value of the slope of the circle became greater than 1. But this adds computations and checking to the algorithm.

One way to eliminate the unequal spacing associated with Eq. 3–17 is to calculate points along the circular boundary using polar coordinates. Expressing the circle equation in parametric polar form yields the pair of equations

$$\begin{aligned} x &= xc + r \cdot \cos\theta \\ y &= yc + r \cdot \sin\theta \end{aligned} \tag{3–18}$$

When a display is generated with these equations using a fixed angular value for θ, a circle is plotted with equally spaced points along the circumference. The step size chosen for θ depends on the application. For circle generation with a raster-scan system, we can set the step size at $1/r$ and calculate closely spaced pixel positions. This step size gives us pixels that are approximately one unit apart.

An improvement in these methods can be made by taking advantage of the symmetry of circles. A given point on the circumference can be mapped into several other circle points by interchanging coordinates and alternating the sign of the coordinate values. As illustrated in Fig. 3–15, a point at position (x, y) on a one-eighth circle sector can be used to plot the other seven points shown. Using this approach, we could generate all pixel positions around a circle for a raster display by calculating only the points within the sector from $x = 0$ to $x = y$.

Determining pixel positions along a circle circumference using either Eq. 3–17 or Eq. 3–18 requires a good deal of computation time. The Pythagorean theorem approach involves multiplications and square root calculations, while the parametric equations contain multiplications and trigonometric calculations. We can improve the efficiency of circle generation by using a method that reduces the computations as much as possible to integer arithmetic.

Bresenham's Circle Algorithm

As in the line-generating algorithm, integer positions along a circular path can be obtained by determining which of two pixels is nearer the circle at each step. To simplify the algorithm statements, we first consider a circle centered at the coordinate origin ($xc = 0$ and $yc = 0$). We also calculate the points for a one-eighth circle segment, assuming that we are going to get the remaining points by symmetry for storage in a raster. (A random-scan system with a vector generator could extend the calculations through one complete cycle.) Unit steps are taken in the x direction, starting from $x = 0$ and ending when $x = y$. The starting coordinate in our algorithm is then $(0, r)$.

The situation at some arbitrary step in the algorithm is shown in Fig. 3–16. We assume that position (x_i, y_i) has been determined to be closer to the circle path. The next position is then either $(x_i + 1, y_i)$ or $(x_i + 1, y_i - 1)$.

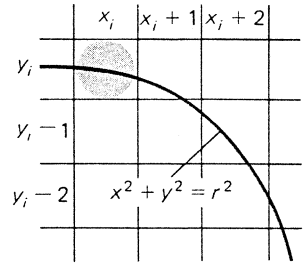


FIGURE 3–16
Section of the screen grid where a circle passing through the point (x_i, y_i) is to be displayed.

From Eq. 3–16, the actual y value on the circle path is determined as

$$y^2 = r^2 - (x_i + 1)^2$$

Figure 3–17 illustrates the relationship between y and the integer coordinate values, y_i and $y_i - 1$. A measure of the difference in coordinate positions can be defined in terms of the square of the y values as

$$\begin{aligned} d_1 &= y_i^2 - y^2 \\ &= y_i^2 - r^2 + (x_i + 1)^2 \end{aligned} \tag{3–19}$$

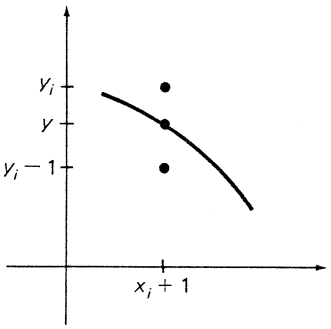


FIGURE 3–17
Coordinate differences between pixel centers and the y position on a circle at $x_i + 1$.

and

$$\begin{aligned} d_2 &= y^2 - (y_i - 1)^2 \\ &= r^2 - (x_i + 1)^2 - (y_i - 1)^2 \end{aligned} \tag{3–20}$$

We now set up a parameter for determining the next coordinate position as the difference between d_1 and d_2:

$$\begin{aligned} p_i &= d_1 - d_2 \\ &= 2(x_i + 1)^2 + y_i^2 + (y_i - 1)^2 - 2r^2 \end{aligned} \tag{3–21}$$

If p_i is negative, we select the pixel at position y_i. Otherwise, we select the pixel at location $y_i - 1$.

The test for selecting the next pixel holds whether the actual path passes above y_i or below $y_i - 1$, as shown in Fig. 3–18. For the first case, Fig. 3–18 (a),

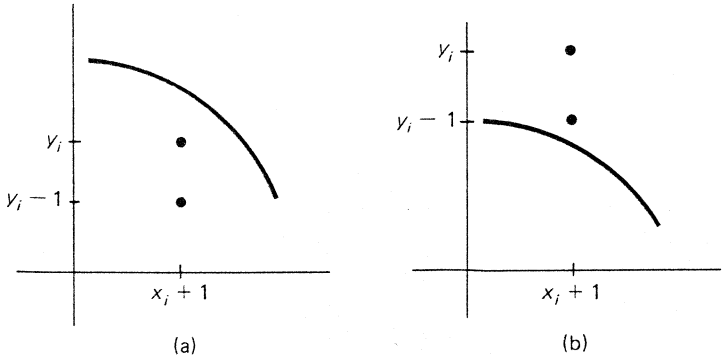


FIGURE 3–18
Possible pixel positions: (a) Both pixel centers are below the circle path, and (b) both pixel centers are above the circle path.

we have $d_1 < 0$, $d_2 > 0$, and $p_i < 0$, so that the point at y_i would be selected. In the second case, Fig. 3–18 (b), $d_1 > 0$ and $d_2 < 0$. Now $p_i > 0$, and the point at $y_i - 1$ is selected.

A recursive form for the parameter p is obtained by evaluating p_{i+1} in terms of p_i:

$$p_{i+1} = 2[(x_i + 1) + 1]^2 + y_{i+1}^2 + (y_{i+1} - 1)^2 - 2r^2$$

This expression can be written in terms of Eq. 3–21 as

$$p_{i+1} = p_i + 4x_i + 6 + 2(y_{i+1}^2 - y_i^2) - 2(y_{i+1} - y_i) \tag{3–22}$$

The y position y_{i+1} is either the same as y_i or the same as $y_i - 1$, depending on the value of p_i. Starting from p_1, the algorithm determines each successive p parameter from the preceding one. We obtain p_1 by setting $(x_1, y_1) = (0, r)$ in Eq. 3–21:

$$p_1 = 3 - 2r \tag{3–23}$$

Figure 3–19 summarizes the steps for calculating integer coordinates closest to the defined circle. To generalize the algorithm so that a circle with an arbitrary center position can be plotted, we simply add xc to each successive x value and add yc to each calculated y value.

FIGURE 3–19
Bresenham's circle algorithm.

1. Select the first position for display as
$$(x_1, y_1) = (0, r)$$
2. Calculate the first parameter as
$$p_1 = 3 - 2r$$
If $p_1 < 0$, the next position is $(x_1 + 1, y_1)$. Otherwise, the next position is $(x_1 + 1, y_1 - 1)$.
3. Continue to increment the x coordinate by unit steps, and calculate each succeeding parameter p from the preceding one. If for the previous parameter we found that $p_i < 0$, then
$$p_{i+1} = p_i + 4x_i + 6.$$
Otherwise (for $p_i \geq 0$),
$$p_{i+1} = p_i + 4(x_i - y_i) + 10$$
Then, if $p_{i+1} < 0$, the next point selected is $(x_i + 2, y_{i+1})$. Otherwise, the next point is $(x_i + 2, y_{i+1} - 1)$. The y coordinate is $y_{i+1} = y_i$, if $p_i < 0$ or $y_{i+1} = y_i - 1$, if $p_i \geq 0$.
4. Repeat the procedures in step 3 until the x and y coordinates are equal.

Although a multiplication is required in the calculation of each parameter, the multiplier is a power of 2, so the multiplication can be implemented as a logical shift operation. All other operations are simply integer additions or subtractions. The following procedure is a coding for this circle algorithm. Input to the procedure are the coordinates for the circle center and the radius. The procedure loads the frame buffer array with points along the circle circumference by calls to the *set_pixel* operation.

```
procedure bres_circle (x_center, y_center, radius : integer);
  var
    p, x, y : integer;
```

```
procedure plot_circle_points;
  begin
    set_pixel (x_center + x, y_center + y);
    set_pixel (x_center - x, y_center + y);
    set_pixel (x_center + x, y_center - y);
    set_pixel (x_center - x, y_center - y);
    set_pixel (x_center + y, y_center + x);
    set_pixel (x_center - y, y_center + x);
    set_pixel (x_center + y, y_center - x);
    set_pixel (x_center - y, y_center - x)
  end;  {plot_circle_points}

begin  {bres_circle}
  x := 0;
  y := radius;
  p := 3 - 2 * radius;
  while x < y do begin
      plot_circle_points;
      if p < 0 then p := p + 4 * x + 6
      else begin
          p := p + 4 * (x - y) + 10;
          y := y - 1
        end;  {if p not < 0}
      x := x + 1
    end;  {while x < y}
    if x = y then plot_circle_points
end;  {bres_circle}
```

Ellipses

A circle-drawing algorithm can be extended to plot either circles or ellipses. In Fig. 3–20 we show one orientation for an ellipse, with r_1 labeling the semimajor axis and r_2 as the semiminor axis. The standard form for the elliptical equation is

$$\left(\frac{x - xc}{r_1}\right)^2 + \left(\frac{y - yc}{r_2}\right)^2 = 1 \tag{3–24}$$

Using polar coordinates r and θ, we can also write the elliptical equations in parametric form:

$$\begin{aligned} x &= xc + r_1 \cdot \cos\theta \\ y &= yc + r_2 \cdot \sin\theta \end{aligned} \tag{3–25}$$

Bresenham's algorithm can be modified to generate elliptical shapes by using Eq. 3–24, instead of the circle equation, in the evaluation of parameter p_i. That is, for an ellipse centered at the origin, we can express y values in the form

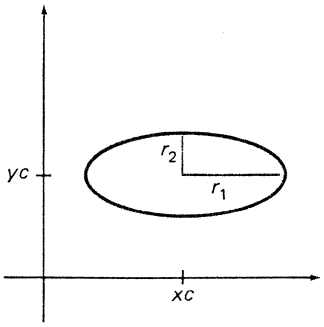
$$y^2 = r_2^2\left(1 - \frac{x^2}{r_1^2}\right) \tag{3–26}$$

The only difference in the algorithm is in the form of the p parameters. An ellipse is plotted at an arbitrary position by adding offsets to the output x and y values, as in the generation of circle positions.

To provide users with the capability for generating ellipses (and circles), a graphics package could include a command of the form

```
ellipse (xc, yc, r1, r2)
```

FIGURE 3–20
Ellipse centered at *(xc, yc)* with semimajor axis r_1 and semiminor axis r_2.



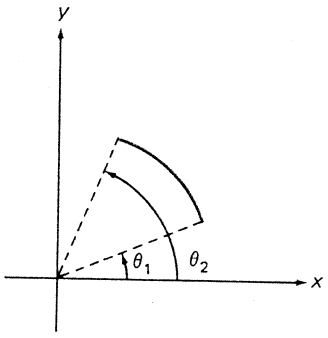


FIGURE 3–21
Circular arc specified by beginning and ending angles. Circle enter is at the coordinate origin.

Coordinates for the ellipse center are assigned to parameters *xc* and *yc*, and the semimajor and semiminor axes are specified in *r*1 and *r*2. If the values assigned to *r*1 and *r*2 are equal, the system displays a circle. These parameters are passed to the ellipse-generating algorithm, which plots the specified figure.

Commands to generate circles and ellipses often include the capability of drawing curve sections by specifying parameters for the line endpoints. Expanding the parameter list allows specification of the beginning and ending angular values for an arc, as illustrated in Fig. 3–21.

3–7 Other Curves

Procedures to display various curves use methods similar to those that generate circles and ellipses. Commonly encountered curves include sine functions, exponential functions, polynomials, probability distributions, and spline functions.

If we can express a curve in functional form, $y = f(x)$, values for y can be calculated and plotted over a specified interval for x. The coordinate values must be rounded to the nearest integer, and the curve path can be filled in with either individual points or straight line segments. For most applications, curves approximated with line segments are more convenient. A point-plotting method leaves gaps in the curve in areas where the magnitude of the slope is greater than 1. To avoid the gaps with a point-plotting method means that we must obtain the inverse function, $x = f^{-1}(y)$, and calculate values of x for given y values whenever the magnitude of the slope becomes large.

Symmetry considerations improve the efficiency of some curve-generating algorithms. Many curves have repeated patterns, so it may be possible to obtain more than one point on the curve with a single calculation. Parabolas and the normal probability distribution are symmetric about a center point, while all points within one cycle of a sine curve can be generated from the points in a 90° interval.

For a curve defined by a data set of discrete coordinate points, we must graph the curve in other ways. One method is simply to plot the individual data points and connect them with straight line segments. Another approach is to use curve-fitting techniques to obtain a curve approximation to the data points. Special curve-fitting methods have been devised for design applications, and we return to this topic in Chapter 10.

FIGURE 3–22
Possible bit pattern for the letter *B*, using an 8 by 8 rectangular grid.

1	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
0	1	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	0
1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0

3–8 Character Generation

Rectangular-grid patterns are typically used to define and plot characters. Figure 3–22 illustrates a bit pattern for the letter *B*, defined on an 8 by 8 dot matrix for use with a bilevel raster system. When this pattern is copied to some area of the frame buffer, the 1 bits designate which pixel positions are to be displayed on the monitor. Rectangular grids for character definitions vary from about 5 by 7 to 9 by 14 or more and are used with both raster and vector systems, although some vector systems generate characters with line segments. Standard character patterns for letters, numbers, and other symbols are predefined and stored in read-only memory. With some systems, additional user-defined character patterns can be accommodated, allowing specialized fonts.

In addition to allowing users to define special characters, graphics packages

Pie charts are used to show the percentage contribution of individual parts to the whole. The next procedure constructs a pie chart, with the number and relative size of the slices determined by input. A sample output from this procedure appears in Fig. 3–25.

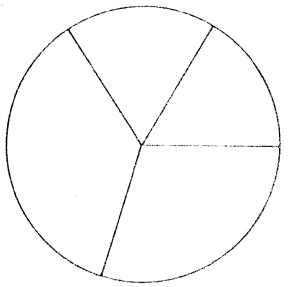


FIGURE 3–25
Output generated from procedure pie_chart.

```
const
  max_slice = 8;
type
  data = array [1..max_slice] of integer;

procedure pie_chart(xc,yc,radius,slices:integer; data_values:data);
  var
    total, k, : integer;
    last_slice, new_slice : real;
  begin
    ellipse (xc, yc, radius, radius);
    total := 0;
    for k := 1 to slices do total := total + data_values[k];
    x[1] := xc;        {every slice line will start at pie center}
    y[1] := yc;
    last_slice := 0;
    for k := 1 to slices do begin
        new_slice := 6.28 * data_values[k] / total + last_slice;
        x[2] := round(xc + radius * cos(new_slice));
        y[2] := round(yc + radius * sin(new_slice));
        polyline (2, x, y);
        last_slice := new_slice    {update last slice}
      end
  end;  {pie_chart}
```

Some variations on the circle equations are output by this next procedure. The shapes shown in Fig. 3–26 are generated by varying the radius r of a circle. Depending on how we vary r, we can produce a spiral, cardioid, limacon, or other similar figure.

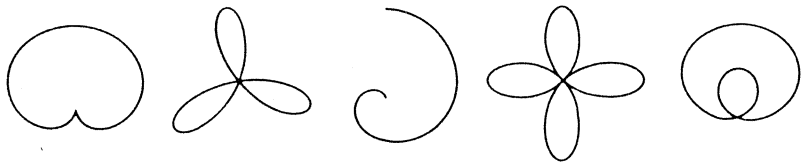


FIGURE 3–26
Curved figures produced with procedure draw_shape.

```
type
  shape = (cardioid, three_leaf, spiral, four_leaf, limacon);

procedure draw_shape (figure : shape; xc, yc, a, b : integer);
  {xc, yc is figure center. a and b are used to compute radius}
  var
    theta, dtheta, r : real;
```

```
      SUBROUTINE CIRCLE (P1, P2, P3, P4, RADIUS, MODE)
C
C     THIS SUBROUTINE WILL DRAW CIRCLES OR CIRCULAR ARCS
C
C     THE MEANINGS OF PARAMETERS P1 - P4 WILL VARY DEPENDING ON
C     THE VALUE OF MODE
C
      INCLUDE 'HCBS.COM/NOLIST'
      REAL XINT(8),YINT(8),AINT(8),X2INT(8),Y2INT(8)
      LOGICAL CCW
      DATA DMAX /32767.0/
C
      CCW=.TRUE.
      IMODE=IABS(MODE)
      IF (IMODE .LE. 0 .OR. IMODE .GT. 7)STOP 'CIRCLE -- ILLEGAL MODE'
      IF (IMODE .EQ. 1) GOTO 100
      R=RADIUS
      R2=R * XSTP
      IF (R2 .GT. DMAX) STOP 'CIRCLE -- RADIUS TOO LARGE'
      IF (R2 .LE. 0.0)STOP 'CIRCLE -- RADIUS TOO SMALL'
      GOTO (100, 200, 200, 400, 500, 600, 700),IMODE
C
100   XS=P1                                    !STARTING POINT OF ARC
      YS=P2                                    !STARTING POINT OF ARC
      XC=P3                                    !CENTER OF CIRCLE
      YC=P4                                    !CENTER OF CIRCLE
      DXC=XS - XC
      DYC=YS - YC
      R=SQRT(DXC**2 + DYC**2)
      R2=R * XSTP
      IF (R2 .GT. DMAX) STOP 'CIRCLE -- RADIUS TOO LARGE'
      IF (R2 .LE. 0.0)STOP 'CIRCLE -- RADIUS TOO SMALL'
      IF (ABS(DYC) .GT. R)DYC=SIGN(R,DYC)
      RAS=ASIN(DYC / R)
      IF (MODE .LT. 0)CCW=.FALSE.
      GOTO 1350
C
200   XS=P1                                    !STARTING POINT OF ARC
      YS=P2                                    !STARTING POINT OF ARC
      XE=P3                                    !ENDING POINT OF ARC
      YE=P4                                    !ENDING POINT OF ARC
      XM=(XS-XE) / 2 + XE
      YM=(YS-YE) / 2 + YE
      SDIST=(XS-XM)**2 + (YS-YM)**2
      PDIST=R**2 - SDIST
      SLOPE=0.0
      IF (YS .NE. YE) SLOPE=- (XS-XE) / (YS-YE)
      IF (PDIST .GE. 0) GOTO 270               !CHECK FOR TOO SHORT
      R=SQRT(SDIST)
      XC=XM
      YC=YM
      GOTO 290
C
270   SQ=SQRT(PDIST / (SLOPE**2 + 1))
      IF (YS-YE) 273, 280, 274
273   IF (MODE .EQ. -2 .OR. MODE .EQ. 3)GOTO 275
      GOTO 278
274   IF (MODE .NE. 2 .AND. MODE .NE. -3)GOTO 278
275   XC=SQ + XM
      GOTO 279
278   XC=-SQ + XM
279   YC=(XC-XM) * SLOPE + YM
      GOTO 290
C
```

```
C                SPECIAL CASE WHERE CHORD IS HORIZONTAL
C
280     XC=XM
        IF (XS-XE) 283, 9000, 284
283     IF (MODE .EQ. -2 .OR. MODE .EQ. 3)GOTO 285
        GOTO 288
284     IF (MODE .NE. 2 .AND. MODE .NE. -3)GOTO 288
285     YC=YM - SQ
        GOTO 290
288     YC=YM + SQ
C
290     DXC=XS - XC
        DYC=YS - YC
        DXE=XE - XC
        DYE=YE - YC
        IF (ABS(DYC) .GT. R)DYC=SIGN(R,DYC)
        RAS=ASIN(DYC / R)
        IF (DXC .LT. 0.0) RAS=3.14159 - RAS
        IF (ABS(DYE) .GT. R)DYE=SIGN(R,DYE)
        RAE=ASIN(DYE / R)
        IF (DXE .LT. 0.0) RAE=3.14159 - RAE
        IF (MODE .LT. 0)CCW=.NOT. CCW
        GOTO 1300
C
400     XS=P1                                   !STARTING POINT OF ARC
        YS=P2                                   !ENDING POINT OF ARC
        RAS=P3  * 0.017453                      !STARTING ANGLE
        DXC=R * COS(RAS)
        DYC=R * SIN(RAS)
        XC=XS - DXC
        YC=YS - DYC
        IF (MODE .LT. 0)CCW=.FALSE.
        GOTO 1350
C
500     XS=P1                                   !STARTING POINT OF ARC
        YS=P2                                   !ENDING POINT OF ARC
        RAS=P3 * 0.017453                       !STARTING ANGLE
        RAE=P4 * 0.017453                       !ENDING ANGLE
        DXC=R * COS(RAS)
        DYC=R * SIN(RAS)
        XC=XS - DXC
        YC=YS - DYC
        DXE=R * COS(RAE)
        DYE=R * SIN(RAE)
        XE=XC + DXE
        YE=YC + DYE
        ARCANG=P4 - P3
        IF (ABS(ARCANG) .LT. 360.0) GOTO 510
        IMODE=4
        GOTO 400
510     IF (ARCANG) 520,9000,1300
520     CCW=.FALSE.
        GOTO 1300
C
600     XC=P1                                   !CENTER OF CIRCLE
        YC=P2                                   !CENTER OF CIRCLE
        RAS=P3 * 0.017453                       !STARTING ANGLE
        DXC=R * COS(RAS)
        DYC=R * SIN(RAS)
        XS=XC + DXC
        YS=YC + DYC
        IF (MODE .LT. 0)CCW=.FALSE.
        GOTO 1350
C
```

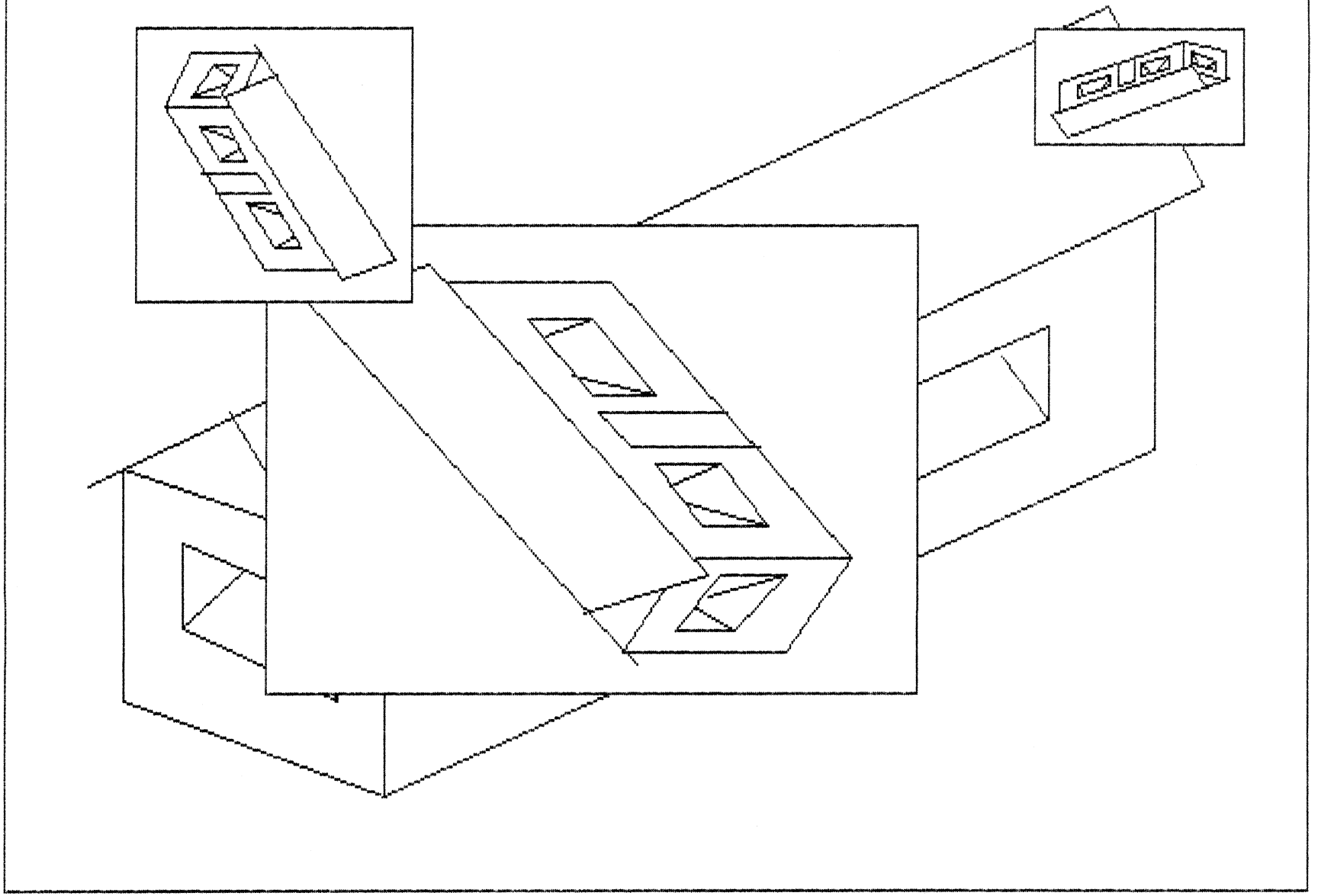
```
700     XC=P1                                           !CENTER OF CIRCLE
        YC=P2                                           !CENTER OF CIRCLE
        RAS=P3 * 0.017453                               !STARTING ANGLE
        RAE=P4 * 0.017453                               !ENDING ANGLE
        DXC=R * COS(RAS)
        DYC=R * SIN(RAS)
        XS=XC + DXC
        YS=YC + DYC
        DXE=R * COS(RAE)
        DYE=R * SIN(RAE)
        XE=XC + DXE
        YE=YC + DYE
        ARCANG=P4 - P3
        IF (ABS(ARCANG) .LT. 360.0) GOTO 710
        IMODE=6
        GOTO 600
710     IF (ARCANG) 720,9000,1300
720     CCW=.FALSE.
C
C                REDUCE ANGLES TO 0 -  6.28318 RADIALS
C
1300    RAE=MOD (RAE,6.28318)
        IF (RAE .LT. 0.0) RAE=RAE + 6.28318
1350    RAS=MOD (RAS,6.28318)
        IF (RAS .LT. 0.0) RAS=RAS + 6.28318
C
C   COMPUTE INTERSECTIONS WITH PAPER LIMITS
C
        ITEL=0
        XXC=XC * XSTP + X0X
        YYC=YC * XSTP + Y0Y
        IF (ABS(PSIZE - YYC) .GE. R2)GOTO 1410
        A=SQRT(R2*R2 - (PSIZE-YYC) * (PSIZE-YYC))
        IF (XXC + A .LT. 0.0)GOTO 1410
        ITEL=ITEL + 1
        XINT(ITEL)=XXC + A
        YINT(ITEL)=PSIZE
        B=ASIN((PSIZE - YYC)/R2)
        IF ( B .LT. 0.0) B=B + 6.28318
        AINT(ITEL)=B
        IF (XXC - A .LT. 0.0)GOTO 1410
        ITEL=ITEL + 1
        XINT(ITEL)=XXC - A
        YINT(ITEL)=PSIZE
        B=3.14159 - B
        IF ( B .LT. 0.0) B=B + 6.28318
        AINT(ITEL)=B
1410    IF (ABS(XXC) .GE. R2)GOTO 1430
        A=SQRT(R2*R2 - XXC * XXC)
        IF (YYC + A .LT. 0.0)GOTO 1430
        IF (YYC + A .GT. PSIZE)GOTO 1420
        ITEL=ITEL + 1
        XINT(ITEL)=0.0
        YINT(ITEL)=YYC + A
        AINT(ITEL)=3.14159 - ACOS(XXC/R2)
1420    IF (YYC - A .LT. 0.0)GOTO 1430
        IF (YYC - A .GT. PSIZE)GOTO 1430
        ITEL=ITEL + 1
        XINT(ITEL)=0.0
        YINT(ITEL)=YYC - A
        AINT(ITEL)=ACOS(XXC/R2) + 3.14159
1430    IF (ABS(YYC) .GE. R2)GOTO 1450
        A=SQRT(R2*R2 - YYC*YYC)
        IF (XXC - A .LT. 0.0)GOTO 1440
```

```
        ITEL=ITEL + 1
        XINT(ITEL)=XXC - A
        YINT(ITEL)=0.0
        AINT(ITEL)=ASIN(YYC/R2) + 3.14159
1440    IF (XXC + A .LT. 0.0)GOTO 1450
        ITEL=ITEL + 1
        XINT(ITEL)=XXC + A
        YINT(ITEL)=0.0
        B=6.28318 - ASIN(YYC/R2)
        IF ( B .GE. 6.28318) B=B - 6.28318
        AINT(ITEL)=B
C
C
1450    XXS=XS * XSTP + X0X
        YYS=YS * XSTP + Y0Y
C
C                COMPUTE NO. OF STEPS FROM START TO CENTER
C
        DXC=-DXC * XSTP
        DYC=-DYC * XSTP
        IDXC=DXC + SIGN(0.5, DXC)
        IDYC=DYC + SIGN(0.5, DYC)
        IF (IDXC .EQ. 0 .AND. IDYC .EQ. 0) GOTO 9000
        CALL VECT(XXS,YYS,3)
        IF (ITEL .NE. 0)GOTO 1500
        IF (XXS .LT. 0.0 .OR. YYS .LT. 0.0 .OR. YYS .GT. PSIZE)GOTO 9000
C
C
        GOTO (3000, 1500, 1500, 3000, 1500, 3000, 1500),IMODE
C
C   INSERT START AND END POINT IN INTERSECTIONS ARRAY
C
1500    ITEL=ITEL +1
        XINT(ITEL)=XXS
        YINT(ITEL)=YYS
        AINT(ITEL)=RAS
        ISTART=ITEL
        IEND=ITEL
        GOTO (1520, 1510, 1510, 1520, 1510, 1520, 1510),IMODE
1510    XXE=XE * XSTP + X0X
        YYE=YE * XSTP + Y0Y
        ITEL=ITEL + 1
        XINT(ITEL)=XXE
        YINT(ITEL)=YYE
        AINT(ITEL)=RAE
        IEND=ITEL
C
C   SORT INTERSECTIONS ARRAY
C
1520    DO 1540 K=ITEL,1,-1
        B=0.0
        J=0
        DO 1530 I=1,ITEL
        IF (AINT(I) .LT. B) GOTO 1530
        J=I
        B=AINT(I)
1530    CONTINUE
        IF (J .EQ. ISTART) ISTART2=K
        IF (J .EQ. IEND) IEND2=K
        X2INT(K)=XINT(J)
        Y2INT(K)=YINT(J)
        AINT(J)=-1.0
1540    CONTINUE
C
```

```
C
C
        K=ISTART2
        KMD=0
        IF (CCW) KMD=1
        ISW=0
        INULSW=0
        XS=X2INT(K)
        YS=Y2INT(K)
        IF (XS .LT. 0.0 .OR. YS .LT. 0.0 .OR. YS .GT. PSIZE) ISW=1
        IF (XS.EQ.0.0 .AND. ((YS.GT.YYC .AND. CCW)
     #                  .OR. (YS.LT.YYC .AND..NOT.CCW)))ISW=1
        IF (YS.EQ.0.0 .AND. ((XS.GT.XXC .AND..NOT. CCW)
     #                  .OR. (XS.LT.XXC .AND.CCW)))ISW=1
        IF (YS.EQ.PSIZE .AND. ((XS.GT.XXC .AND. CCW)
     #                   .OR. (XS.LT.XXC .AND..NOT.CCW)))ISW=1
C
1550    IF (CCW) K=K + 1
        IF (.NOT. CCW) K=K - 1
        IF (K .NE. 0) GOTO 1560
        INULSW=1
        K=ITEL
1560    IF (K .LE. ITEL) GOTO 1570
        INULSW=1
        K=1
1570    XE=X2INT(K)
        YE=Y2INT(K)
        DXE=XE - XXC
        DYE=YE - YYC
        IDXE=DXE + SIGN(0.5, DXE)
        IDYE=DYE + SIGN(0.5, DYE)
        IDX=IDXC + IDXE
        IDY=IDYC + IDYE
        IF (IDX .EQ. 0 .AND. IDY .EQ. 0)GOTO 1620
        IF (ISW .NE. 0) GOTO 1610
        CALL BUFF(16, 0, 6)
        CALL BUFF(12, 1, 0)
        CALL BUFF(KMD, 1, 0)
        CALL BUFF(IDXC, IDYC, 2)
        CALL BUFF(IDXE, IDYE, 2)
C
C                   ADJUST PEN POSITION
C
        XAX=XXC + IDXE
        YAY=YYC + IDYE
        XBX=XXC + IDXE
        YBY=YYC + IDYE
        XMAX=MAX(XMAX,XAX)
        IOP=3
        GOTO 1620
1610    CALL VECT(XE,YE,3)
C
1620    ISW=IEOR(ISW,1)
        IF (K .EQ. IEND2)GOTO 1700
        IDXC=-IDXE
        IDYC=-IDYE
        GOTO 1550
C
C   READY WITH ARC'S
C   ADJUST XMAX
C
1700    IF (INULSW .EQ. 0)GOTO 9000
        B=XXC + R2
        XMAX=MAX(XMAX,B)
```

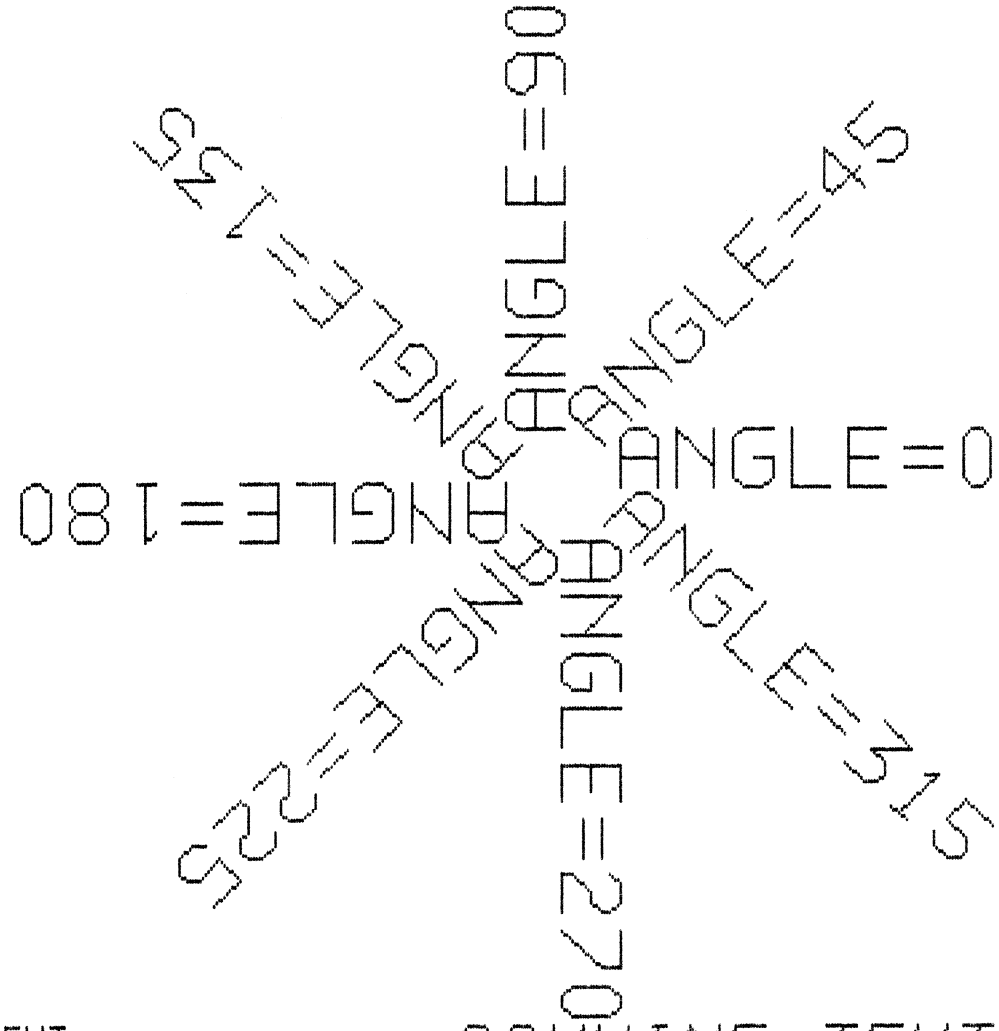
```
        GOTO 9000
C
C         GENERATE FULL CIRCLE COMMAND
C
3000    KMD=4
        IF (CCW) KMD=5
        CALL BUFF(9, 0, 6)
        CALL BUFF(12, 1, 0)
        CALL BUFF(KMD, 1, 0)
        CALL BUFF(IDXC, IDYC, 2)
        XMAX=MAX(XMAX,XXC+R2)
C
C         ALL DONE
C
9000    RETURN
        END
```

STRIKE ENTER FOR NEXT PAGE



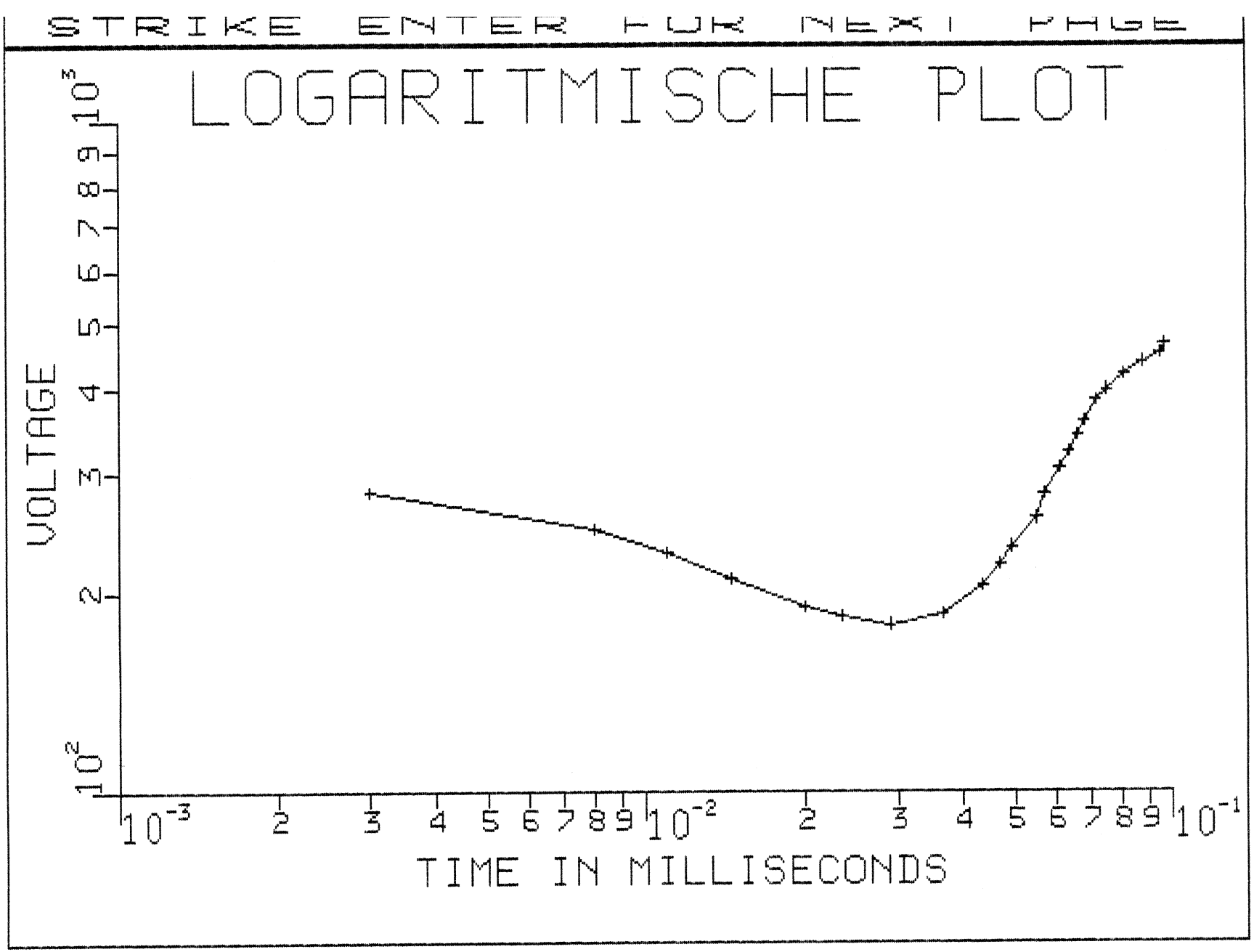
STRIKE ENTER FOR NEXT PAGE

BREDE TEXT



SMALLE TEXT

SCHUINE TEXT


STRIKE ENTER FOR NEXT PAGE
LOGARITMISCHE PLOT
VOLTAGE
10^2
2
3
4
5
6
7
8
9
10^3
10^{-3}
2
3
4
5
6
7
8
9
10^{-2}
2
3
4
5
6
7
8
9
10^{-1}
TIME IN MILLISECONDS

STRIKE ENTER FOR NEXT PAGE

Ook letters in bitmap 5 x 7 Font

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdef9hijklmnoP9rstuvwxyz

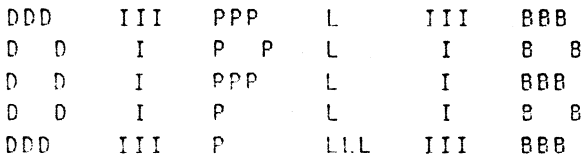
0123456789!@#$%^&*( )<>( )[]?

En in hoge resolutie met een 7 x 14 Font

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz

0123456789!@#$%^&*( )<>{}-[ ]?



This library is a Device Independent Plot LIBrary (DIPLIB). This means that any user who has written a plotprogram for one of the available graphical devices, can run his program on another device without making any modification to his program.

Most of the CALCOMP calls can be used as described in the manuals. Some routines have been changed or added as described in this text. So if you are writing a program, first check this text for the routine you want to use, otherwise use the description in the CALCOMP manuals.

Due to the introduction of the "WINDOW" and "VIEWPORT" concept the routine "FACTOR" has been deleted.

The library is linked to your program by adding "SYS$LIBRARY:DIPLIB/LIB" to your link command.

INTRODUCTION

The library makes use of three different Cartesian coordinate systems:

1. World coordinates

2. Normalized device coordinates

3. Physical device coordinates

1 WORLD COORDINATES

Graphical world coordinates are device independent Cartesian coordinates defined by your application program to describe locations and sizes. You can adjust the graphical world to whatever size is the most convenient.

The mapping of world coordinates onto the screen or plotter is handled by the plotpacket.

World coordinates can represent any unit of measure. The only requirement imposed by the plotpacket is that world coordinates must be supplied as real (floating point) numbers.

1.1 Windows

The WINDOW subroutine defines the window, which is a rectangular portion of world coordinate space that is currently used by your program. You provide the lower and upper bounds of both dimensions of the window.

There is NO DEFAULT value for the window, so a call to this subroutine is mandatory.

The plotpacket clips the world coordinate objects so that the portions that would fall outside the window are removed. This is called WINDOW CLIPPING.

2 NORMALIZED DEVICE COORDINATES (NDC'S)

Normalized device coordinates are device independent coordinates defined by the plotpacket to describe the screen or paper. In other words, normalized device coordinates are a device independent way of talking about a physical device. The mapping of normalized device coordinates onto the screen or paper is handled entirely by the plotpacket.

Normalized device coordinates are real numbers in the range 0 to 1. Thus the entire screen or paper has the bounds 0<=X<=1 and 0<=Y<=1.

2.1 Viewports

Your program can use all of normalized device coordinate space or any rectangular portion of it. The portion used by your program is called the viewport. The VIEWPORT subroutine specifies the exact bounds of the viewport. It would seem that normalized device coordinate space should be square (since both sides are one unit long) but, in reality, it's a rectangle. Since NDC space describes the screen or paper, the actual shape of NDC space depends on the shape of the screen or paper.

Thus, if you use the full viewport, you will find that your images are somewhat distorted. There are two simple ways to correct this:

1. Adjust the window to compensate for the aspect ratio of the viewport.

2. Adjust the viewport to compensate for the aspect ratio of the device.

It is also possible to rotate the window over a multiple of 90 degrees before mapping it onto the viewport.

2.2 Default Settings

When a call to the subroutine VIEWPORT is omitted, the following values are assumed:

1. For terminals the window will be mapped onto the entire screen (viewport coordinates 0,0 and 1,1).

2. For plotters the window coordinates will be interpreted as cm. The viewport size is adjusted automatically.

3 PHYSICAL DEVICE COORDINATES

Physical device coordinates are device dependent coordinates for specifying positions on the device. Physical device coordinates are not available for the user.

	Physical device coordinates		Paper size (cm)	
	X	Y	X	Y
VT125 terminal	0- 767	479- 0		
VT240 terminal	0- 799	479- 0		
HP terminal	0- 719	0- 359		
TEKTRONIX terminal	0- 1023	0- 767		
Calcomp plotter (small)	0-30000	0- 5800	150.00	29.00
Calcomp plotter (large)	0-30000	0-22000	150.00	110.00
HP plotter (A4)	0-10603	0- 7721	26.38	19.21
HP plotter (A3)	0-15370	0-10602	38.23	26.37

PLOTS

The PLOTS subroutine is used to initialize the plotpacket for a particular device. It must be called once for each device to initialize. Then the WINDOW subroutine has to be called before any other plot call is made.

It is possible to call PLOTS more than once, in that case the plot output will be directed to the initialized devices simultaneously although they may have different viewports.

The calling sequence has three arguments:

```
CALL PLOTS (IDEV,ISIZ,ILUN)
```

IDEV = Type of device to initialize
- 1 = Disk file
- 2 = VT240 terminal
- 3 = HP terminal
- 4 = Calcomp plotter
- 5 = HP plotter
- 6 = VT125 terminal
- 7 = TEKTRONIX terminal

ISIZ = Paper size used (only for plotters)
- even = small paper (A4 for the HP plotter)
- odd = large paper (A3 for the HP plotter)

ILUN = The absolute value of this parameter specifies the Logical unit number to use for this device. If a terminal is addressed this lun is not used for the actual physical output, but it must be a unique number to be able to distinquish the devices from each other. If for the Calcomp plotter this parameter is < 0, no banner line will be produced.

PLOTEX

This subroutine ends the plotting on the specified device. If the Calcomp plotter was addressed, the plot information is send to the plotter queue. If the file was addressed, the file will be closed. After this call no plot output will be send to the device.

The calling sequence has one argument:

```
CALL PLOTEX (LUN)
```

The absolute value of LUN is used to specify the device. If LUN<0 and the Calcomp plotter was specified, no plot will be made. If LUN<0 and the file was specified, the file will be closed and deleted.

WINDOW

The WINDOW subroutine is used to define the user's world coordinates. It also resets the viewport for the plotters so that the world parameters are interpreted as cm. It does not affect the viewport for the terminals.

WINDOW has to be called before any plot call is made and may be called again as often as desired.

The calling sequence has four arguments:

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

The arguments define the lower and upper limits of the window.

VIEWPORT

The VIEWPORT subroutine defines the portion of the screen or the paper to be used for the plot. The coordinates are expressed in normalized device coordinates (0 to 1 for the entire screen or paper). A call to VIEWPORT is optional and must be done with great care if the calcomp plotter is used. A viewport coordinate of 1 in the upper X limit for this plotter means a plot of 150 cm in length.

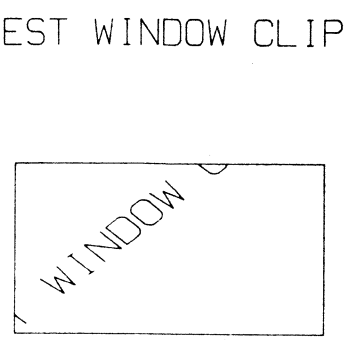
The calling sequence has six arguments:

```
CALL VIEWPORT (ILUN,X1,X2,Y1,Y2,IROT)
```

ILUN = The absolute value of this parameter defines the Logical unit number of the device on which the viewport has to be defined. If ILUN < 0 only the rotation parameter if significant (the others will be ignored).
X1,Y1= Lower limit of the viewport in NDC's
X2,Y2= Upper limit of the viewport in NDC's
IROT = Number of multiples of 90 degrees the window has to be rotated before mapping onto the viewport.

Example:

```
   CALL PLOTS(5,0,1)
   CALL WINDOW(0.,30.,0.,20.)
C
   CALL SYMBOL(0.,18.,1.,'TEST WINDOW CLIPPING',0.,20)
   CALL WINDOW(5.,15.,5.,15.)
   CALL VIEWPORT(1,0.05,0.1167,0.10,0.44,0)
   CALL PLOT(5.,5.,3)
   CALL PLOT(15.,5.,2)
   CALL PLOT(15.,15.,2)
   CALL PLOT(5.,15.,2)
   CALL PLOT(5.,5.,2)
   CALL SYMBOL(3.,3.,1.,'TEST WINDOW CLIPPING',45.,20)
C
   CALL PLOT(30.,0.,999)
   END
```



GROFF and GRON

These two subroutines are used to switch the terminal from graphic mode to alphanumeric mode (GROFF) and visa versa (GRON). This switch enables the user to perform any interactive terminal I/O and then continue the plot. The routines may be called for the plotters, but don't have any effect.

The calling sequences have no arguments:

```
CALL GROFF
CALL GRON
```

ERASE

This subroutine erases the screen of the terminal and/or rewinds the output file. For the Calcomp plotter this means that all previous data is lost. It has no effect on the HP plotter.

The calling sequence has no arguments:

```
CALL ERASE
```

REDRAW

This subroutine redraws the entire plot. It can only be called if file output was initialized.

The calling sequence has one argument:

```
CALL REDRAW (LUN)
```

LUN is a logical unit number, temporarely used for copying the data from the plotfile.

REDRES

This subroutine changes the resolution used for drawing arc's. The normal resolution is multiplied by this factor.

CALL REDRES (FACT)

FACT is the multiplication factor to be used.

CHGHPS

This subroutine changes the plotting speed of the HP plotter. This is nessecary when plotting on transparent film.

CALL CHGHPS

TSLANT

This subroutine changes the text slant-angle. The default is 90 degrees.

CALL TSLANT (angle)

NEWSET

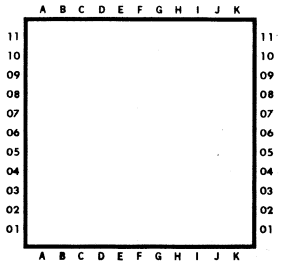
This subroutine specifies the character set to be used. Currently there are 5 different character sets (0 through 4). The default is 0. Each of the characters of these sets may be slanted (see TSLANT).

Set 3 is a proportional character set, the others are not. This means that the width of the characters depends on the character itself (an "i" will be smaller than an "m").

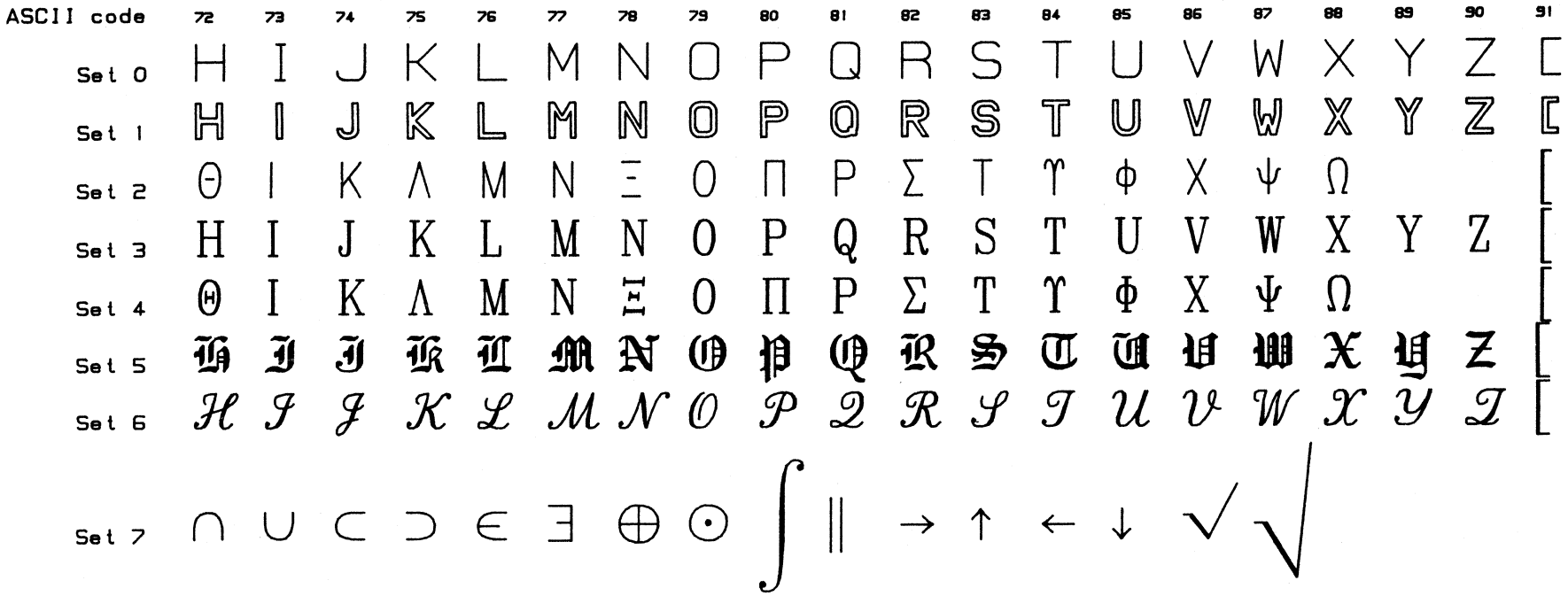
The calling sequence has one argument:

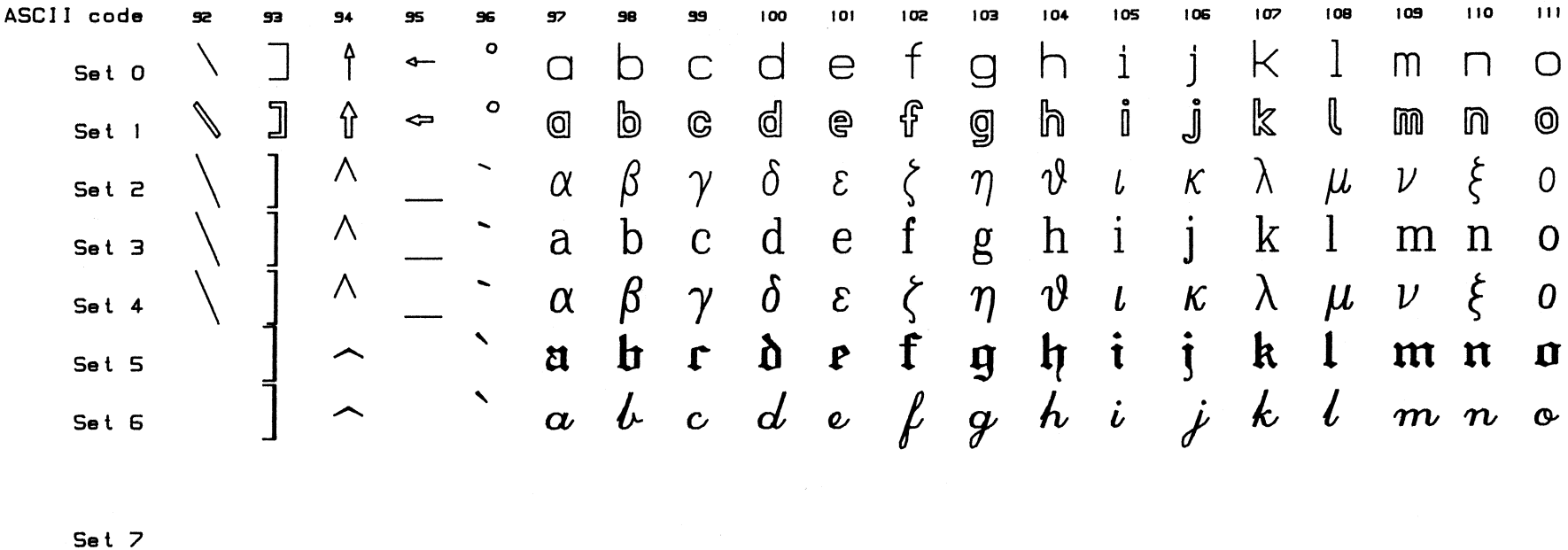
CALL NEWSET (ISET)

ISET is the set number to be selected.




ASCII code
32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51
Set 0
Set 1
Set 2
Set 3
Set 4
Set 5
Set 6
Set 7
ASCII code
52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
Set 0
Set 1
Set 2
Set 3
Set 4
Set 5
Set 6
Set 7


ASCII code
72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91
Set 0
Set 1
Set 2
Set 3
Set 4
Set 5
Set 6
Set 7


ASCII code
92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111
Set 0
Set 1
Set 2
Set 3
Set 4
Set 5
Set 6
Set 7


ASCII code
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
Set 0
Set 1
Set 2
Set 3
Set 4
Set 5
Set 6
Set 7

REDRES

This subroutine changes the resolution used for drawing arc's. The normal resolution is multiplied by this factor.

CALL REDRES (FACT)

FACT is the multiplication factor to be used.

CHGHPS

This subroutine changes the plotting speed of the HP plotter. This is nessecary when plotting on transparent film.

CALL CHGHPS

TSLANT

This subroutine changes the text slant-angle. The default is 90 degrees.

CALL TSLANT (angle)

AXIS2 and SCALE2

The normal scale and axis subroutines divide the axis into intervals of exactly 1 cm. This means that the graph plotted with the subroutine LINE on this axis will not use the entire axis. To avoid this problem a second set of scale and axis subroutines is made, which varies the intervals along the axis so that the graph uses the entire axis. The calling sequences are the same as for the normal calls exept for the name.

CALL SCALE2 (ARRAY, AXLEN, NPTS, INC)

CALL AXIS2(XPAGE,YPAGE,IBCD,NCHAR,AXLEN,ANGLE,FIRSTV,DELTAV)

AXIS3

This is a special version of the AXIS subroutine which allows the caller to specify all the axis parameters in the subroutine call. This gives us the following possibilities:

- Change the height of the numbers along the axis (normal = 0.21)
- Change the height of the title of the axis (normal = 0.28)
- Display numbers along the axis with a different number of cyphers in the fractional part (normal = 2).
- Get a different tick-length (normal = 0.178).
- Get a different tick-distance (normal = 1.0).

```
CALL AXIS3 (XPAGE,YPAGE,TITLE,NCHAR,HEIGH,AXLEN,ANGLE,
            TDIST,TLENG,FIRST,DELTA,INCRE,THGHT,NDECS)
```

XPAGE, YPAGE = The coordinates of the axis line's starting point.

TITLE = The title of the axis, which is centered and placed parallel to the axis line.

NCHAR = The absolute value specifies the number of characters in the axis title. If the sign is positive, all annotation appears on the positive (counterclockwise) side of the axis (normal for the Y-axis).

HEIGH = The height of the title of the axis. If the height is zero, no title is plotted.

AXLEN = The length of the axis line.

ANGLE = The angle in degrees at which the axis is drawn.

TDIST = The distance between two ticks on the axis.

TLENG = The length of each tick.

FIRST = The value appearing at the first tick.

DELTA = The number of data units per tick.

INCRE = The absolute value specifies the number of ticks between two data values to be displayed along the axis. If the sign is negative, the ticks without data value are shortened by a factor 1.25.

THGHT = The height of the data values displayed. If the height is zero, no data values are plotted.

NDECS = The number of decimals in the fractional part of the data values. If NDECS <= 0 no fractional part is displayed including the decimal point (giving an integer value).

International Division

International Headquarters
2411 W. La Palma Avenue
Anaheim, California 92801
(714) 821-2011
Telex 910-591-1154

European Headquarters
CalComp Europe Ltd.
75/77 High Street
Tunbridge Wells
Kent TN1 1X2
United Kingdom
Phone 0892-41522
Telex 95409 CCPTW G

Americas and Pacific Headquarters
2411 W. La Palma Avenue
[illegible]heim, California 92801
[illegible]ne (714) 821-2686
[illegible]elex 910-591-1154

Middle East and Africa
CalComp
Louizis Riankour No. 64/14B3
Athens 616, Greece
Phone 01/6925011
Telex 210614 CCPA GR

CalComp Subsidiaries

Belgium
N.V. CalComp S.A.
Rue duSaphirstraat 29, B9
1040 Brussels, Belgium
Phone 7363980
Telex 23913

Canada
CalComp Canada
55 Westmore Drive
Rexdale, Ontario M9V 3Y6
Phone (416) 745-9610
Telex 06989222

CalComp Canada
100 Alexis Nihon Blvd.
Suite 875
St. Laurent, Quebec H4M 2T4
Phone (514) 744-6455
Telex 05 82 6652

CalComp Canada
Building B-3
2616 16th St. N.E.
Calgary, Alberta T2E TI8
Phone (403) 230-1178

CalComp Canada Ltd.
120 Highway 15, Suite 206
Ottawa, Ontario K2H 5Z1
Phone (613) 820-8463

France
CalComp S.A.
43 rue de la Breche-aux-Loups
Paris 75012 France
Phone 344-1507
Telex 680684

Germany (FRG)
CalComp GmbH
Werfstrasse 37
4000 Düsseldorf 11, F.R.G.
Phone 501193
Telex 8584661

Elmshorner Strasse 7-11
2080 Pinneberg
Phone 2 50 85
Telex 2 189 063

Neusser Strasse 9
8000 München 40
Phone 36 30 94
Telex 522-929

Schumannstrasse 163
6050 Offenbach
Phone 83 57 71
Telex 4 152 652

Italy
CalComp S.p.A.
Palazzo F1
20094 Milanofiori Assago (M1)
Phone (02) 8242001
Telex 312360

Viale Masini, 20
40126 Bologna, Italy
Phone (051) 352540/366900

Via Thailandia, 27
00144 Roma - Eur, Italy
Phone (06) 5914905/5914405

Japan
CalComp Pacific, Inc.
Azabu Building
3-5-27, Roppongi
Minato-ku, Tokyo 106
Phone 03/585-7101/02/03
Telex 126242

CalComp Pacific, Inc.
Daini Yuyama Bldg., 7th Flr.
3-19-13 Nishinakajima
Yodogawa-ku, Osaka
Phone 06/304/2012

Mexico
CalComp S.A. de C.V.
Plateros No. 7-206
Mexico 19, D.F.
Phone (905) 524-0914
Telex (temporary) 01771400

Netherlands
CalComp B.V.
Maalderij 21
P.O. Box 444
Amstelveen, The Netherlands
Phone 457-351
Telex 12599

Switzerland
CalComp GmbH
Buelhofstrasse 360
8185 Winkel bei Buelach
Switzerland
Phone 8607935
Telex 59252

United Kingdom
CalComp Ltd.
Cory House
The Ring
Bracknell, Berks U.K. RG12 1ER
Phone Bracknell 50211
Telex 848949

CalComp Europe Ltd.
(Technical Services)
4 The Courtyard, Denmark Street
Wokingham, Berks. U.K. RG11 2AZ
Phone 0734-781500
Telex 847258 CCPWOK G

(Training Support)
Crown House, Marconi Road
London Road
Newbury, Berks. U.K. RG13 2DH
Phone 0635-32400
Telex 847630 CCPETS G

CalComp Distributors

For a complete list of CalComp Distributors in 30 foreign countries, ask for a copy of the International Sales Offices brochure.

CALCOMP
INTERNATIONAL DIVISION

California Computer Products, Inc., International Division, 2411 W. La Palma Ave., Anaheim, CA 92801
Telephone (714) 821-2011 TWX 910-591-1154

10255-901-017-1/ID CCP 4C 881 Printed in U.S.A.

```
      SUBROUTINE AXISC(XPAGE,YPAGE,IBCD,NC,SIZE,THETA,YMIN,DY)
      CHARACTER*3 IM(12)
      CHARACTER*1 IBCD(*)
      INTEGER*2 IYB,I
      DATA IM/'JAN','FEB','MAR','APR','MAY','JUN','JUL'
     $    ,'AUG','SEP','OCT','NOV','DEC'/
      RH=.3
      IF(1./DY .LT. 3.*RH)RH=1./DY/3.5
      IF(NC .GE. 0)THEN
        SIG=1.0
       ELSE
        SIG=-1.0
        ENDIF
      NAC=IABS(NC)
      TH=THETA*0.0174533292
      STH=SIN(TH)
      CTH=COS(TH)
      XO=XPAGE
      YO=YPAGE
C...EERSTE LIJNSTUK
      CALL PLOT(XO,YO,3)
      IF(AINT(YMIN) .NE. YMIN)THEN
        IYB=INT(YMIN)+1
       ELSE
        IYB=INT(YMIN)
        ENDIF
      X=(FLOAT(IYB)-YMIN)/DY
      XN=XO+X*CTH
      YN=YO+X*STH
      CALL PLOT(XN,YN,2)
      CALL PLOT(XO+X*CTH-RH*SIG*STH,YO+X*STH+RH*SIG*CTH,2)
C...LOOP
      DO 20 I=IYB,nINT(YMIN+SIZE*DY)
        CALL PLOT(XN,YN,3)
        X=(FLOAT(I)-YMIN)/DY
        XN=XO+X*CTH
        YN=YO+X*STH
        CALL PLOT(XN,YN,2)
        CALL PLOT(XO+X*CTH-RH*SIG*STH,YO+X*STH+RH*SIG*CTH,2)
        XL=(FLOAT(I)-0.5-YMIN)/DY-1.5*RH
        IF(NC .GT. 0)THEN
          YL=RH
         ELSE
          YL=-2.*RH
          ENDIF
        CALL SYMBOL(XO+XL*CTH-YL*STH,YO+XL*STH+YL*CTH
     $    ,RH,IM(1+MOD(I-2,12)),THETA,3)
        IF(1+MOD(I-2,12) .EQ. 1)THEN
          IF(NC .GT. 0)THEN
            YL=2.5*RH
           ELSE
            YL=-3.5*RH
            ENDIF
          CALL NUMBER(XO+XL*CTH-YL*STH,YO+XL*STH+YL*CTH
     $      ,RH,I/12,0,THETA,-1)
          ENDIF
20    CONTINUE
C...AFSLUITING
```

```
      CALL PLOT(XN,YN,3)
      X=SIZE
      XN=XO+X*CTH
      YN=YO+X*STH
      CALL PLOT(XN,YN,2)
C...LABEL
      RH=0.3556
      XL=SIZE/2.-RH*NAC
      IF(NC .GT. 0)THEN
        YL=2.2*RH
       ELSE
        YL=-3.2*RH
        ENDIF
      CALL SYMBOL(XO+XL*CTH-YL*STH,YO+XL*STH+YL*CTH
     $  ,RH,IBCD,THETA,NAC)
      RETURN
      END
```

```
      SUBROUTINE SCALE2(ARRAY,AXLEN,NPTS,INC)
C...  ARRAY   NAME OF ARRAY CONTAINING VALUES TO BE SCALED.
C...  AXLEN   LENGTH IN INCHES OVER WHICH ARRAY IS TO BE SCALED.
C...  NPTS    NUMBER OF POINTS TO BE SCALED.
C...  INC     INCREMENT OF LOCATION OF SUCCESSIVE POINTS.
      INTEGER*2 I,K,N
      DIMENSION ARRAY(*)
      K=IABS(INC)
      N=NPTS*K
      YMIN=ARRAY(1)
      YMAX=YMIN
      DO 10 I=1,N,K
        YMIN=AMIN1(YMIN,ARRAY(I))
10      YMAX=AMAX1(YMAX,ARRAY(I))
      DELTA=(YMAX-YMIN)/AXLEN
      IF (DELTA .EQ. 0.0)GOTO 20
      ARRAY(N+1)=YMIN
      ARRAY(N+1+K)=DELTA
      RETURN
20    IF(YMIN)40,50,30
30    ARRAY(N+1)=0.0
      ARRAY(N+1+K)=(2.*YMIN)/AXLEN
      RETURN
40    ARRAY(N+1)=2.*YMIN
      ARRAY(N+1+K)=(-2.*YMIN)/AXLEN
      RETURN
50    ARRAY(N+1)=-1.
      ARRAY(N+1+K)=2./AXLEN
      RETURN
      END
```

```
      subroutine axis(
     $  xpage,ypage,ibcd,nchar,axlen,angle,firstv,deltav)
c...     xpage,ypage  coordinates of starting point of axis, in cm
c...     ibcd    axis title.
c...     nchar   number of characters in title. + for c.c-w side.
c...     axlen   floating point axis length in cm.
c...     angle   angle of axis from the x-direction, in degrees.
c...     firstv  scale value at the first tic mark.
c...     deltav  change in scale between tic marks one cm.  apart
      integer*2 ntic,ntc,i
      character*1 ibcd(*)
      kn=nchar
      a=1.0
      if(kn)1,2,2
1     a=-a
      kn=-kn
2     ex=0.0
      adx=abs(deltav)
      if(adx)3,7,3
3     if(adx-99.0)6,4,4
4     adx=adx/10.0
      ex=ex+1.0
      goto 3
5     adx=adx*10.0
      ex=ex-1.0
6     if(adx-0.01)5,7,7
7     xval=firstv*10.0**(-ex)
      adx=2.0*deltav*10.0**(-ex)
      sth=angle*0.0174533
      cth=cos(sth)
      sth=sin(sth)
      cth2=cth+cth
      sth2=sth+sth
      dxb=-0.254
      dyb=0.38*a-0.127
      xn=xpage+dxb*cth-dyb*sth
      yn=ypage+dyb*cth+dxb*sth
      ntic=axlen+1.0
      ntc=(ntic+1)/2
      nt=ntc/2
      do 20 i=1,ntc
      call number(xn,yn,0.210,xval,angle,2)
      if(i-1)10,10,105
10    xn=xpage+2.0*dxb*cth-dyb*sth
      yn=ypage+2.0*dxb*sth+dyb*cth
105   xval=xval+adx
      xn=xn+cth2
      yn=yn+sth2
      if(nt)20,11,20
11    z=kn
      if(ex)12,13,12
12    z=z+7.0
13    dxb=-.14*z+axlen*0.5
      dyb=0.825*a-0.190
      xt=xpage+dxb*cth-dyb*sth
      yt=ypage+dyb*cth+dxb*sth
      call symbol(xt,yt,0.28,ibcd,angle,kn)
      if(ex)14,20,14
14    z=kn+2
      xt=xt+z*cth*0.28
      yt=yt+z*sth*0.28
      call symbol(xt,yt,0.28,'*10',angle,3)
      xt=xt+(3.0*cth-0.8*sth)*0.28
      yt=yt+(3.0*sth+0.8*cth)*0.28
      call number(xt,yt,0.21,ex,angle,-1)
20    nt=nt-1
      call plot(xpage+axlen*cth,ypage+axlen*sth,3)
      dxb=-.178*a*sth
      dyb=+.178*a*cth
      a=ntic-1
      xn=xpage+a*cth
      yn=ypage+a*sth
      do 30 i=1,ntic
      call plot(xn,yn,2)
      call plot(xn+dxb,yn+dyb,2)
      call plot(xn,yn,2)
      xn=xn-cth
      yn=yn-sth
30    continue
      return
      end
c*************************************************************************
      subroutine axis2(
     $  xpage,ypage,ibcd,nchar,axlen,angle,firstv,deltav)
c...     xpage,ypage  coordinates of starting point of axis, in cm
```

```
c...    ibcd    axis title.
c...    nchar   number of characters in title. + for c.c-w side.
c...    axlen   floating point axis length in cm.
c...    angle   angle of axis from the x-direction, in degrees.
c...    firstv  scale value at the first tic mark.
c...    deltav  change in scale between tic marks one cm.  apart
        character*1 ibcd(*)
        integer*2 i,iex,is,isw
        dimension save(5)
        data save/1.0,2.0,4.0,5.0,8.0/
        if(deltav .eq. 0.0)stop 'axis2 -- devtav=0. not supported'
        a=1.0
        if(nchar .lt. 0)a=-1.0
        sth=angle*0.0174533
        cth=cos(sth)
        sth=sin(sth)
        iex=alog10(deltav)
        p=10.0**iex
        ai=deltav/p
        do 10 is=1,5
          if(save(is).ge. ai)goto 20
10        continue
        is=5
20      delt=save(is)*p
        ainter=(axlen*deltav)/delt
        ai=firstv/delt
        i=ai
        if(float(i).ne. ai .and. firstv .ge. 0.0)i=i+1
        startv=i*delt
        pint=axlen/ainter
        offset=pint*(startv-firstv)/delt
        isw=0
30      dx=-0.254+offset
        dy=0.38*a-0.127
        xn=xpage+dx*cth-dy*sth
        yn=ypage+dx*sth+dy*cth
        if(isw .eq. 0)call number(xn,yn,0.210,startv/p,angle,2)
c       isw=ieor(isw,1)
        if(isw.eq.0)then
          isw=1
         else
          isw=0
          endif
        startv=startv+delt
        offset=offset+pint
        if(offset .le. axlen)goto 30
        if(nchar .eq. 0)goto 40
        z=iabs(nchar)
        if(iex .ne. 0)z=z+7.0
        dx=-0.14*z+axlen*0.5
        dy=0.825*a-0.190
        xn=xpage+dx*cth-dy*sth
        yn=ypage+dx*sth+dy*cth
        call symbol(xn,yn,0.28,ibcd,angle,iabs(nchar))
        if(iex .eq. 0)goto 40
        z=iabs(nchar)+2
        xn=xn+z*cth*0.28
        yn=yn+z*sth*0.28
        call symbol(xn,yn,0.28,'*10',angle,3)
        xn=xn+(3.0*cth-0.8*sth)*0.28
        yn=yn+(3.0*sth+0.8*cth)*0.28
        z=iex
        call number(xn,yn,0.21,z,angle,-1)
40      dx=-0.178*a*sth
        dy=0.178*a*cth
        xn=xpage+axlen*cth
        yn=ypage+axlen*sth
        call plot(xn,yn,3)
50      offset=offset-pint
        if(offset .lt. 0.0)goto 60
        xn=xpage+offset*cth
        yn=ypage+offset*sth
        call plot(xn,yn,2)
        call plot(xn+dx,yn+dy,2)
        call plot(xn,yn,2)
        goto 50
60      call plot(xpage,ypage,2)
        return
        end
c*************************************************************************
c*************************************************************************
        subroutine clipline(x,y,c)
        integer*2 c,x,y
        integer*2 idevice,ixmaxs,iymaxs,irotation
        integer*2 ixvmin,ixvmax,iyvmin,iyvmax
```

```
      integer*2 ipenplotter,maxpenplotter,idashlinetype
      real xrmin,xrmax,yrmin,yrmax
      real xcurrent,ycurrent,rfact
      real xvmin,xvmax,yvmin,yvmax
      real xiscale,xascale,yiscale,yascale
      common /diplib/idevice
     $ ,xrmin,xrmax,yrmin,yrmax
     $ ,ixmaxs,iymaxs
     $ ,xcurrent,ycurrent
     $ ,rfact
     $ ,ixvmin,ixvmax,iyvmin,iyvmax
     $ , xvmin, xvmax, yvmin, yvmax
     $ ,irotation
     $ ,ipenplotter,maxpenplotter
     $ ,xiscale,xascale,yiscale,yascale
     $ ,aspectsymbol,talicsymbol
     $ ,idashlinetype,dashlinelenght
     $ ,ipfile,corx,cory
      c=0
      if(x .lt. ixvmin)then
        c=c+1
       elseif(x .gt. ixvmax)then
        c=c+2
        endif
      if(y .lt. iyvmin)then
        c=c+4
       elseif(y .gt. iyvmax)then
        c=c+8
        endif
      return
      end
c*************************************************************************
      subroutine dashl(x,y,n,k)
c...  xarray  name of array containing abscissas of
c...          data points to be plotted
c...  yarray  name of array containing ordinates of
c...          data points to be plotted
c...  npts    number of data points to be plotted
c...  inc     increment between elements of the array
      call dashs(idum,1)
      call line(x,y,n,k,0,0)
      call dashs(idum,0)
      return
      end
c
c*************************************************************************
c
      subroutine dashp(x,y,dl)
c...  xpage, ypage    are the coordinates of the point to which the dashed
c...                  line is to be drawn.
c...  dash            is the length of the dash and space between dashes.
      real a(2)
      a(1)=-dl
      a(2)=dl
      call dashs(a,2)
      call plot(x,y,2)
      call dashs(a,0)
      return
      end
c*************************************************************************
      subroutine dashs(array,icnt)
      dimension array(*)
      return
      end
c*************************************************************************
      subroutine drawscal
c...  scale for line drawing
      integer*2 idevice,ixmaxs,iymaxs,irotation
      integer*2 ixvmin,ixvmax,iyvmin,iyvmax
      integer*2 ipenplotter,maxpenplotter,idashlinetype
      real xrmin,xrmax,yrmin,yrmax
      real xcurrent,ycurrent,rfact
      real xvmin,xvmax,yvmin,yvmax
      real xiscale,xascale,yiscale,yascale
      common /diplib/idevice
     $ ,xrmin,xrmax,yrmin,yrmax
     $ ,ixmaxs,iymaxs
     $ ,xcurrent,ycurrent
     $ ,rfact
     $ ,ixvmin,ixvmax,iyvmin,iyvmax
     $ , xvmin, xvmax, yvmin, yvmax
     $ ,irotation
     $ ,ipenplotter,maxpenplotter
     $ ,xiscale,xascale,yiscale,yascale
     $ ,aspectsymbol,talicsymbol
```

```
     $  ,idashlinetype,dashlinelenght
     $  ,ipfile,corx,cory
        xvmin=real(ixvmin)
        xvmax=real(ixvmax)
        yvmin=real(iyvmin)
        yvmax=real(iyvmax)
        if(irotation .eq. 0 .or. irotation .eq. 2)then
          amx=(xvmax-xvmin)/(xrmax-xrmin)
          amy=(yvmax-yvmin)/(yrmax-yrmin)
         else
          amx=(xvmax-xvmin)/(yrmax-yrmin)
          amy=(yvmax-yvmin)/(xrmax-xrmin)
          endif
        if(irotation .eq. 0)then
          xiscale=xvmin-xrmin*amx
          xascale= amx
          yiscale=yvmin-yrmin*amy
          yascale= amy
         else if(irotation .eq. 1)then
          xiscale=xvmax+yrmin*amx
          xascale=-amx
          yiscale=yvmin-xrmin*amy
          yascale= amy
         else if(irotation .eq. 2)then
          xiscale=xvmax+xrmin*amx
          xascale=-amx
          yiscale=yvmax+yrmin*amy
          yascale=-amy
         else if(irotation .eq. 3)then
          xiscale=xvmin-yrmin*amx
          xascale= amx
          yiscale=yvmax+xrmin*amy
          yascale=-amy
          endif
        return
        end
c*******************************************************************************
        subroutine erase
c...    clears the viewport
        integer*2 i,jpen
        integer*2 idevice,ixmaxs,iymaxs,irotation
        integer*2 ixvmin,ixvmax,iyvmin,iyvmax
        integer*2 ipenplotter,maxpenplotter,idashlinetype
        real xrmin,xrmax,yrmin,yrmax
        real xcurrent,ycurrent,rfact
        real xvmin,xvmax,yvmin,yvmax
        real xiscale,xascale,yiscale,yascale
        common /diplib/idevice
     $  ,xrmin,xrmax,yrmin,yrmax
     $  ,ixmaxs,iymaxs
     $  ,xcurrent,ycurrent
     $  ,rfact
     $  ,ixvmin,ixvmax,iyvmin,iyvmax
     $  , xvmin, xvmax, yvmin, yvmax
     $  ,irotation
     $  ,ipenplotter,maxpenplotter
     $  ,xiscale,xascale,yiscale,yascale
     $  ,aspectsymbol,talicsymbol
     $  ,idashlinetype,dashlinelenght
     $  ,ipfile,corx,cory
        if(   ipfile .ne. 0
     $  .and. ixvmin .eq. 0
     $  .and. ixvmax .eq. ixmaxs
     $  .and. iyvmin .eq. 0
     $  .and. iyvmax .eq. iymaxs)then
          call plots(idevice,0,0)
         else
          jjpen=ipenplotter
          ipenplotter=0
          do 100 i=iyvmin,iyvmax
100         call showline(ixvmin,i,ixvmax,i)
          ipenplotter=jjpen
          endif
        return
        end
c*******************************************************************************
        subroutine factor(fact)
c...    scale window with fact
        integer*2 idevice,ixmaxs,iymaxs,irotation
        integer*2 ixvmin,ixvmax,iyvmin,iyvmax
        integer*2 ipenplotter,maxpenplotter,idashlinetype
        real xrmin,xrmax,yrmin,yrmax
        real xcurrent,ycurrent,rfact
        real xvmin,xvmax,yvmin,yvmax
        real xiscale,xascale,yiscale,yascale
```

```
      common /diplib/idevice
     $ ,xrmin,xrmax,yrmin,yrmax
     $ ,ixmaxs,iymaxs
     $ ,xcurrent,ycurrent
     $ ,rfact
     $ ,ixvmin,ixvmax,iyvmin,iyvmax
     $ , xvmin, xvmax, yvmin, yvmax
     $ ,irotation
     $ ,ipenplotter,maxpenplotter
     $ ,xiscale,xascale,yiscale,yascale
     $ ,aspectsymbol,talicsymbol
     $ ,idashlinetype,dashlinelenght
     $ ,ipfile,corx,cory
      xrmin=xrmin/fact
      xrmax=xrmax/fact
      yrmin=yrmin/fact
      yrmax=yrmax/fact
      rfact=fact
      call drawscal
      return
      end
c*************************************************************************
      subroutine fit(xa,ya,xb,yb,xc,yc)
      dimension ss(8,9),theta(2)
      integer*2 i,ktra
      m=2
      dy=yc-ya
      dx=xc-xa
      z3=sqrt(dy**2+dx**2)
      if(z3)20,20,21
21    do 8 i=1,2
      if(abs(dx)-abs(dy))1,2,2
1     theta(i)=1.5708-atan(abs(dx/dy))
      goto 3
2     theta(i)=atan(abs(dy/dx))
3     if(dx)25,26,26
25    if(dy)5,4,4
26    if(dy)4,5,5
4     theta(i)=-theta(i)
5     if(dx)6,7,7
6     theta(i)=theta(i)+3.1416
7     dx=xb-xa
8     dy=yb-ya
      z2=sqrt(dy**2+dx**2)*cos(theta(2)-theta(1))
      if(z2)20,20,22
22    ss(1,3)=xa-xc
      ss(2,3)=xa-xb
      ktra=1
      goto 13
16    a=ss(1,3)
      b=ss(2,3)
      ss(1,3)=ya-yc
      ss(2,3)=ya-yb
      ktra=2
      goto 13
17    call where(x,y,fctr)
      dz=0.01/fctr
      z=dz
      call plot(xa,ya,3)
      c=ss(1,3)
      d=ss(2,3)
18    y=d*z+ya
      x=(a*z+b)*z+xa
      call plot(x,y,2)
      z=z+dz
      if(z-z3)18,19,19
19    call plot(xc,yc,2)
      return
13    ss(1,1)=z3*z3
      ss(1,2)=z3
      ss(2,1)=z2*z2
      ss(2,2)=z2
      call solut(ss,m)
      if(m)20,20,14
14    goto(16,17),ktra
20    call plot(xa,ya,3)
      call plot(xb,yb,2)
      goto 19
      end
c*************************************************************************
      subroutine frame
c...  draws a frame around the viewport
      integer*2 idevice,ixmaxs,iymaxs,irotation
      integer*2 ixvmin,ixvmax,iyvmin,iyvmax
      integer*2 ipenplotter,maxpenplotter,idashlinetype
```

```
      real xrmin,xrmax,yrmin,yrmax
      real xcurrent,ycurrent,rfact
      real xvmin,xvmax,yvmin,yvmax
      real xiscale,xascale,yiscale,yascale
      common /diplib/idevice
     $ ,xrmin,xrmax,yrmin,yrmax
     $ ,ixmaxs,iymaxs
     $ ,xcurrent,ycurrent
     $ ,rfact
     $ ,ixvmin,ixvmax,iyvmin,iyvmax
     $ , xvmin, xvmax, yvmin, yvmax
     $ ,irotation
     $ ,ipenplotter,maxpenplotter
     $ ,xiscale,xascale,yiscale,yascale
     $ ,aspectsymbol,talicsymbol
     $ ,idashlinetype,dashlinelenght
     $ ,ipfile,corx,cory
      call where(xsave,ysave,r)
      call plot(xrmin,yrmin,3)
      call plot(xrmin,yrmax,2)
      call plot(xrmax,yrmax,2)
      call plot(xrmax,yrmin,2)
      call plot(xrmin,yrmin,2)
      call plot(xsave,ysave,3)
      return
      end
c
c*************************************************************************
      subroutine grid(x,y,xs,ys,m,n)
c...  where -(x,y) is the starting position of grid
c...  xs    is the space of grid in x direction.
c...  ys    is the space of grid in y direction
c...  m     is the number of division in x direction.
c...  n     is the number of divisions in y direction.
      integer*2 i,im
      y0=y
      im=n+1
      xf=x+xs*float(m)
      x0=x
      call plot(x,y,3)
      do 10 i=1,im
      call plot(x0,y0,2)
      call plot(xf,y0,2)
      y0=y0+ys
      xt=xf
      xf=x0
10    x0=xt
      x0=x
      y0=y
      xf=y+ys*float(n)
      im=m+1
      do 20 i=1,im
      call plot(x0,xf,2)
      call plot(x0,y0,2)
      x0=x0+xs
      xt=xf
      xf=y0
20    y0=xt
      return
      end
c*************************************************************************
      subroutine groff
c     grafisch --> alfanumeriek
      integer*2 idevice,ixmaxs,iymaxs,irotation
      integer*2 ixvmin,ixvmax,iyvmin,iyvmax
      integer*2 ipenplotter,maxpenplotter,idashlinetype
      real xrmin,xrmax,yrmin,yrmax
      real xcurrent,ycurrent,rfact
      real xvmin,xvmax,yvmin,yvmax
      real xiscale,xascale,yiscale,yascale
      common /diplib/idevice
     $ ,xrmin,xrmax,yrmin,yrmax
     $ ,ixmaxs,iymaxs
     $ ,xcurrent,ycurrent
     $ ,rfact
     $ ,ixvmin,ixvmax,iyvmin,iyvmax
     $ , xvmin, xvmax, yvmin, yvmax
     $ ,irotation
     $ ,ipenplotter,maxpenplotter
     $ ,xiscale,xascale,yiscale,yascale
     $ ,aspectsymbol,talicsymbol
     $ ,idashlinetype,dashlinelenght
     $ ,ipfile,corx,cory
      if(ipfile .ne. 0)then
        close(unit=99)
```

```
          endif
        call int10(3,0,0)
        return
        end
c*************************************************************************
        subroutine line(xarray,yarray,npts,inc,lintyp,inteq)
c...    xarray  name of array containing abscissa or x values.
c...    yarray  name of array containing ordinate or y values.
c...    npts    number of points to be plotted.
c...    inc     increment of location of successive points.
c...    lintyp  control type of line--symbols, line, or combination.
c...    inteq   integer equivalent of symbol to be used, if any.
        dimension xarray(*),yarray(*)
        integer*2 i,icodea,ipena,kk,ldx,lmin,na,nf,nl,nt
        lmin=npts*inc+1
        ldx=lmin+inc
        nl=lmin-inc
        firstx=xarray(lmin)
        deltax=xarray(ldx)
        firsty=yarray(lmin)
        deltay=yarray(ldx)
        call where(xn,yn,df)
        df=amax1(abs((xarray(1)-firstx)/deltax-xn),
     $   abs((yarray(1)-firsty)/deltay-yn))
        dl=amax1(abs((xarray(nl)-firstx)/deltax-xn),
     $   abs((yarray(nl)-firsty)/deltay-yn))
        ipen=3
        icode=-1
        nt=iabs(lintyp)
        if(lintyp)7,6,7
6       nt=1
7       if(df-dl)9,9,8
8       nf=nl
        na=((npts-1)/nt)*nt+nt-(npts-1)
        kk=-inc
        goto 10
9       nf=1
        na=nt
        kk=inc
10      if(lintyp)11,12,13
11      ipena=3
        icodea=-1
        lsw=1
        goto 15
12      na=ldx
13      ipena=2
        icodea=-2
        lsw=0
15      do 30 i=1,npts
        xn=(xarray(nf)-firstx)/deltax
        yn=(yarray(nf)-firsty)/deltay
        if(na-nt)20,21,22
20      if(lsw)23,22,23
21      call symbol(xn,yn,0.20,char(inteq),0.0,icode)
        na=1
        goto 25
22      call plot(xn,yn,ipen)
23      na=na+1
25      nf=nf+kk
        icode=icodea
30      ipen=ipena
        return
        end
c*************************************************************************
        subroutine maxpen(ipen)
c...    ipen becomes the maximum available pen
        integer*2 idevice,ixmaxs,iymaxs,irotation
        integer*2 ixvmin,ixvmax,iyvmin,iyvmax
        integer*2 ipenplotter,maxpenplotter,idashlinetype
        real xrmin,xrmax,yrmin,yrmax
        real xcurrent,ycurrent,rfact
        real xvmin,xvmax,yvmin,yvmax
        real xiscale,xascale,yiscale,yascale
        common /diplib/idevice
     $  ,xrmin,xrmax,yrmin,yrmax
     $  ,ixmaxs,iymaxs
     $  ,xcurrent,ycurrent
     $  ,rfact
     $  ,ixvmin,ixvmax,iyvmin,iyvmax
     $  , xvmin, xvmax, yvmin, yvmax
     $  ,irotation
     $  ,ipenplotter,maxpenplotter
     $  ,xiscale,xascale,yiscale,yascale
     $  ,aspectsymbol,talicsymbol
     $  ,idashlinetype,dashlinelenght
```

```
     $  ,ipfile,corx,cory
        ipen=maxpenplotter
        ipenplotter=ipen
        return
        end
c*************************************************************************
        subroutine newpen(ipen)
c...    define a new pen
        integer*2 idevice,ixmaxs,iymaxs,irotation
        integer*2 ixvmin,ixvmax,iyvmin,iyvmax
        integer*2 ipenplotter,maxpenplotter,idashlinetype
        real xrmin,xrmax,yrmin,yrmax
        real xcurrent,ycurrent,rfact
        real xvmin,xvmax,yvmin,yvmax
        real xiscale,xascale,yiscale,yascale
        common /diplib/idevice
     $  ,xrmin,xrmax,yrmin,yrmax
     $  ,ixmaxs,iymaxs
     $  ,xcurrent,ycurrent
     $  ,rfact
     $  ,ixvmin,ixvmax,iyvmin,iyvmax
     $  , xvmin, xvmax, yvmin, yvmax
     $  ,irotation
     $  ,ipenplotter,maxpenplotter
     $  ,xiscale,xascale,yiscale,yascale
     $  ,aspectsymbol,talicsymbol
     $  ,idashlinetype,dashlinelenght
     $  ,ipfile,corx,cory
        if(ipen.ge.0 .and. ipen.le.maxpenplotter)then
          ipenplotter=ipen
         else
c..clip pen nummer
          ipen=maxpenplotter
          endif
        return
        end
c*************************************************************************
        subroutine number(xpage,ypage,height,fpn,angle,ndec)
c...    xpage,ypage coordinates of lower left corner of number.
c...    height  height of plotted number.
c...    fpn     floating point number to be plotted.
c...    angle   angle at which number is plotted, in degrees.
c...    ndec    number of decimal places to be drawn.
c...    this version of number requires the symbol version with
c...    999. x, y feature, and  nc=0 feature.
        integer*2 minus,izero,ipoint,i,j,k,n,ilp,mn,maxn
        character num(20)
        data minus/45/,izero/48/,ipoint/46/
        ii=0
        fpv=fpn
        n=ndec
        maxn=9
        if(n-maxn)11,11,10
10      n=maxn
11      if(n+maxn)12,20,20
12      n=-maxn
20      if(fpv)21,30,30
21      ii=ii+1
        num(ii)=char(minus)
30      mn=-n
        if(n)31,32,32
31      mn=mn-1
32      fpv=abs(fpv)+(0.5*10.**mn)
        i=alog10(fpv)+1.0
        ilp=i
        if(n+1)40,41,41
40      ilp=ilp+n+1
41      if(ilp)50,50, 51
50      ii=ii+1
        num(ii)=char(izero)
        goto 61
51      if(ilp+n-18)54,54,52
52      n=-1
        if(ilp-19)54,54,53
53      ilp=19
54      do 60 j=1,ilp
        k=fpv*10.**(j-i)
        if(k-9)57,57,55
55      k=9
57      ii=ii+1
        num(ii)=char(k+izero)
        fpv=fpv-(float(k)*10.**(i-j))
60      continue
61      if(n)99,70,70
70      ii=ii+1
```

```
        num(ii)=char(ipoint)
        if(n)99,99,80
80      do 90 j=1,n
        k=fpv*10.
        if(k-9)84,84,82
82      k=9
84      ii=ii+1
        num(ii)=char(k+izero)
90      fpv=fpv*10.-float(k)
99      call symbol(xpage,ypage,height,num,angle,ii)
        return
        end
c*******************************************************************************
        subroutine plot(xpage,ypage,ipen)
c...    move the pen in a straight line
c...    from the current location to xpage,ypage
c...    ipen= 2 pen down
c...    ipen= 3 pen up
c...    ipen=-2 pen down and define new origin
c...    ipen=-3 pen up   and define new origin
c...    ipen=999 ends plotting
        real xpage,ypage
        integer*2 x1,x2,y1,y2,iret
        integer*2 c1,c2,c
        integer*4 x,y
        character niks
        integer*2 idevice,ixmaxs,iymaxs,irotation
        integer*2 ixvmin,ixvmax,iyvmin,iyvmax
        integer*2 ipenplotter,maxpenplotter,idashlinetype
        real xrmin,xrmax,yrmin,yrmax
        real xcurrent,ycurrent,rfact
        real xvmin,xvmax,yvmin,yvmax
        real xiscale,xascale,yiscale,yascale
        common /diplib/idevice
     $  ,xrmin,xrmax,yrmin,yrmax
     $  ,ixmaxs,iymaxs
     $  ,xcurrent,ycurrent
     $  ,rfact
     $  ,ixvmin,ixvmax,iyvmin,iyvmax
     $  , xvmin, xvmax, yvmin, yvmax
     $  ,irotation
     $  ,ipenplotter,maxpenplotter
     $  ,xiscale,xascale,yiscale,yascale
     $  ,aspectsymbol,talicsymbol
     $  ,idashlinetype,dashlinelenght
     $  ,ipfile,corx,cory
c...    einde plot
        if(iabs(ipen).eq. 999)then
          read(*,20)niks
20        format(a1)
          call groff
          return
          endif
c...    draw line
        if(iabs(ipen) .eq. 2)then
          if(irotation .eq. 0 .or. irotation .eq. 2)then
            x1=nint(xiscale+xascale*xcurrent)
            y1=nint(yiscale+yascale*ycurrent)
            x2=nint(xiscale+xascale*xpage   )
            y2=nint(yiscale+yascale*ypage   )
           else
            x1=nint(xiscale+xascale*ycurrent)
            y1=nint(yiscale+yascale*xcurrent)
            x2=nint(xiscale+xascale*ypage   )
            y2=nint(yiscale+yascale*xpage   )
            endif
          if(x1.lt.ixvmin .or. x1.gt.ixvmax
     $  .or. x2.lt.ixvmin .or. x2.gt.ixvmax
     $  .or. y1.lt.iyvmin .or. y1.gt.iyvmax
     $  .or. y2.lt.iyvmin .or. y2.gt.iyvmax)then
              call clipline(x1,y1,c1)
              call clipline(x2,y2,c2)
1000          if(c1 .ne. 0 .or. c2 .ne. 0)then
c               if(iand(c1,c2).ne.0)goto 200
                if(mod(c1  ,2).eq.1 .and. mod(c2  ,2).eq.1)goto 200
                if(mod(c1/2,2).eq.1 .and. mod(c2/2,2).eq.1)goto 200
                if(mod(c1/4,2).eq.1 .and. mod(c2/4,2).eq.1)goto 200
                if(mod(c1/8,2).eq.1 .and. mod(c2/8,2).eq.1)goto 200
                c=c1
                if(c.eq.0)c=c2
c...      links
                if(mod(c,2).ne.0)then
                  y=y1+(y2-y1)*(ixvmin-x1)/(x2-x1)
                  x=ixvmin
c...      rechts
```

```
               elseif(mod(c/2,2).ne.0)then
                y=y1+(y2-y1)*(ixvmax-x1)/(x2-x1)
                x=ixvmax
c...    onder
               elseif(mod(c/4,2).ne.0)then
                x=x1+(x2-x1)*(iyvmin-y1)/(y2-y1)
                y=iyvmin
c...    boven
               elseif(mod(c/8,2).ne.0)then
                x=x1+(x2-x1)*(iyvmax-y1)/(y2-y1)
                y=iyvmax
                endif
c
              if(c.eq.c1)then
                x1=x
                y1=y
                call clipline(x1,y1,c1)
               else
                x2=x
                y2=y
                call clipline(x2,y2,c2)
                endif
              goto 1000
              endif
            endif
          call showline(x1,y1,x2,y2)
          endif
c...    update current pen position
200     xcurrent=xpage
        ycurrent=ypage
c...    define new origin
        if(ipen .lt. 0)then
          xrmin=xrmin-xpage
          xrmax=xrmax-xpage
          yrmin=yrmin-ypage
          yrmax=yrmax-ypage
          xcurrent=0.0
          ycurrent=0.0
          call drawscal
          endif
        return
        end
c*************************************************************************
        subroutine plots(idev,isize,ilun)
c...    reset plotting device
c...    idev:
c...    0=640 x 400     Olivetti ATT-6300
c...    1=640 x 200     IBM hires
c...    2=320 x 200     IBM color
c...    3=320 x 200     IBM monochrome
c...    4=720 x 350     Hercules monochrome
c...    5=320 x 200     EGA color
c...    6=640 x 200     EGA color
c...    7=640 x 350     EGA monochrome
c...    8=640 x 350     EGA color
c...    if isize < 0 call for display type --> idev
c...    ilun has no meaning
        character filen(14)
        character*14 fil
        equivalence(filen,fil)
        integer*2 rax,rcx,rdx
        integer*2 idevice,ixmaxs,iymaxs,irotation
        integer*2 ixvmin,ixvmax,iyvmin,iyvmax
        integer*2 ipenplotter,maxpenplotter,idashlinetype
        real xrmin,xrmax,yrmin,yrmax
        real xcurrent,ycurrent,rfact
        real xvmin,xvmax,yvmin,yvmax
        real xiscale,xascale,yiscale,yascale
        common /diplib/idevice
     $  ,xrmin,xrmax,yrmin,yrmax
     $  ,ixmaxs,iymaxs
     $  ,xcurrent,ycurrent
     $  ,rfact
     $  ,ixvmin,ixvmax,iyvmin,iyvmax
     $  , xvmin, xvmax, yvmin, yvmax
     $  ,irotation
     $  ,ipenplotter,maxpenplotter
     $  ,xiscale,xascale,yiscale,yascale
     $  ,aspectsymbol,talicsymbol
     $  ,idashlinetype,dashlinelenght
     $  ,ipfile,corx,cory
c...    maximum implemented devices
        maxdevices      = 9
        xrmin           = 0.0
        xrmax           =29.7
```

```
        yrmin           = 0.0
        yrmax           =20.9
        xcurrent        = 0.0
        ycurrent        = 0.0
        rfact           = 1.0
        irotation       = 0
        ipenplotter     = 1
        aspectsymbol    =1.0
        talicsymbol     = 0.0
1       if(isize .lt. 0)then
5         write(*,10)
10        format(
     $    ' type of device to initialyze, add 100 for printfile',/,
     $    ' 0=Olivetti-M21/M24, ATT-6300     640x400   2 color',/,
     $    ' 1=IBM hires                      640x200   2 color',/,
     $    ' 2=IBM color                      320x200  16 color',/,
     $    ' 3=IBM monochrome                 320x200   2 color',/,
     $    ' 4=Hercules monochrome            720x350   2 color',/,
     $    ' 5=EGA color                      320x200  16 color',/,
     $    ' 6=EGA color                      640x200  16 color',/,
     $    ' 7=EGA monochrome                 640x350   2 color',/,
     $    ' 8=EGA color                      640x350   4 color',/,
     $    ' 9=EGA color                      640x350  16 color',/,
     $    '  =')
          read(*,*)idev
          if(mod(idev,100).lt.0.or.mod(idev,100).gt.maxdevices)goto 5
          end if
c..Geen Hercules
        if(idev .eq. 4)then
          isize=-1
          goto 1
          endif
        idevice=idev
c
c...set resolution  640 x 400 olivetti monochrome
        if(mod(idev,100) .eq. 0)then
          call int10(64,0,0)
          ixmaxs=639
          iymaxs=399
          maxpenplotter=1
c
c...set resolution  640 x 200 ibm monochrome
        else if(mod(idev,100) .eq. 1)then
          call int10(6,0,0)
          ixmaxs=639
          iymaxs=199
          maxpenplotter=1
c
c...set resolution  320 x 200 ibm color
        else if(mod(idev,100) .eq. 2)then
          call int10(4,0,0)
          ixmaxs=319
          iymaxs=199
          maxpenplotter=15
c
c...set resolution  320 x 200 ibm monochrome
        else if(mod(idev,100) .eq. 3)then
          call int10(5,0,0)
          ixmaxs=319
          iymaxs=199
          maxpenplotter=15
c
c...set resolution  720 x 350 hercules monochrome
        else if(mod(idev,100) .eq. 4)then
c...hercules (mode,x,y,icolor)
c...mode=1 set text mode
c...mode=2 set graphics mode
c...mode=0 plot x, y
c...       call hercules(2,0,0,0)
c...       moet getest worden
           ixmaxs=719
           iymaxs=349
           maxpenplotter=1
c
c       EGA 320x200 color
        else if(mod(idev,100) .eq. 5)then
          call int10(13,0,0)
          ixmaxs= 319
          iymaxs= 199
          maxpenplotter=15
c
c       EGA 640x200 color
        else if(mod(idev,100) .eq. 6)then
          call int10(14,0,0)
          ixmaxs= 639
```

```
          iymaxs=199
          maxpenplotter=15
c
c       EGA 640x350 monochrome
        else if(mod(idev,100) .eq. 7)then
          call int10(15,0,0)
          ixmaxs=639
          iymaxs=349
          maxpenplotter=1
c
c       EGA 640x350 color 4 color
        else if(mod(idev,100) .eq. 8)then
          call int10(16,0,0)
          ixmaxs=639
          iymaxs=349
          maxpenplotter=3
c
c       EGA 640x350 color 16 color
        else if(mod(idev,100) .eq. 9)then
          call int10(16,0,0)
          ixmaxs=639
          iymaxs=349
          maxpenplotter=15
c
          end if
c... open print file
        if(idev .ge. 100)then
          fil='print000.dat'
98        write(*,99)
99        format(' File number for print XXX --> printXXX.dat  number = ')
          read(*,*)ip
          if(ipfile.lt.0 .or. ipfile.gt.999)goto 98
          filen(8)=char(48+mod(ipfile    ,10))
          filen(7)=char(48+mod(ipfile/10 ,10))
          filen(6)=char(48+mod(ipfile/100,10))
          open(99,file=filen,status='new',form='unformatted')
200       isave=ixmaxs
          ixmaxs=4319
          corx=real(isave)/real(ixmaxs)
          isave=iymaxs
          iymaxs=1919
          cory=real(isave)/real(iymaxs)
         else
          ipfile=0
          corx=1.
          cory=1.
          endif
        ipenplotter=maxpenplotter
        ixvmin=0
        ixvmax=ixmaxs
        iyvmin=0
        iyvmax=iymaxs
        call drawscal
        return
        end
c*************************************************************************
        subroutine poly(x,y,sl,rn,th)
        integer*2 i,n
        n=rn
        xn=x
        yn=y
        tho=th*0.01745
        call plot(x,y,3)
        if(n)10,100,20
10      th1=-6.2832/rn
        th2=-th1*2.0
        n=-n
        do 11 i=1,n
        xn=xn+sl*cos(tho)
        yn=yn+sl*sin(tho)
        call plot(xn,yn,2)
        tho=tho+th1
        xn=xn+sl*cos(tho)
        yn=yn+sl*sin(tho)
        call plot(xn,yn,2)
11      tho=tho+th2
100     return
20      th1=6.2832/rn
        do 21 i=1,n
        xn=xn+sl*cos(tho)
        yn=yn+sl*sin(tho)
        call plot(xn,yn,2)
21      tho=tho+th1
        return
        end
```

```
c*************************************************************************
      subroutine rect(x,y,h,w,th,iv)
      theta=th/57.2958
      xs=sin(theta)
      xc=cos(theta)
      call plot(x,y,iv)
      x1=x-h*xs
      y1=y+h*xc
      call plot(x1,y1,2)
      x1=x1+w*xc
      y1=y1+w*xs
      call plot(x1,y1,2)
      x1=x+w*xc
      y1=y+w*xs
      call plot(x1,y1,2)
      call plot(x,y,2)
      return
      end
c*************************************************************************
      subroutine scale(array,axlen,npts,inc)
c...  array   name of array containing values to be scaled.
c...  axlen   length in inches over which array is to be scaled.
c...  npts    number of points to be scaled.
c...  inc     increment of location of successive points.
      integer*2 i,k,n,is
      dimension  array(*),save(7)
      data save/1.0,2.0,4.0,5.0,8.0,10.0,20.0/
      fad=0.01
      k=iabs(inc)
      n=npts*k
      y0=array(1)
      yn=y0
      do 25 i=1,n,k
        y0=amin1(y0,array(i))
25      yn=amax1(yn,array(i))
      firstv=y0
      if(y0)34,35,35
34    fad=fad-1.0
35    deltav=(yn-firstv)/axlen
      if(deltav)70,70,40
40    i=alog10(deltav)+1000.0
      p=10.0**(i-1000)
      deltav=deltav/p-0.01
      do 45 i=1,6
        is=i
        if(save(i)-deltav)45,50,50
45      continue
50    deltav=save(is)*p
      firstv=deltav*aint(y0/deltav+fad)
      t=firstv+(axlen+0.01)*deltav
      if(t-yn)55,57,57
55    firstv=p*aint(y0/p+fad)
      t=firstv+(axlen+.01)*deltav
      if(t-yn)56,57,57
56    is=is+1
      goto 50
57    firstv=firstv-aint((axlen+(firstv-yn)/deltav)/2.0)*deltav
      if(y0*firstv)58,58,59
58    firstv=0.0
59    if(inc)61,61,65
61    firstv=firstv+aint(axlen+.5)*deltav
      deltav=-deltav
65    n=n+1
      array(n)=firstv
      n=n+k
      array(n)=deltav
67    return
70    deltav=2.0*firstv
      deltav=abs(deltav/axlen)+1.
      goto 40
      end
c*************************************************************************
      subroutine scale2(array,axlen,npts,inc)
c...  array   name of array containing values to be scaled.
c...  axlen   length in inches over which array is to be scaled.
c...  npts    number of points to be scaled.
c...  inc     increment of location of successive points.
      integer*2 i,k,n
      dimension array(*)
      k=iabs(inc)
      n=npts*k
      ymin=array(1)
      ymax=ymin
      do 10 i=1,n,k
        ymin=amin1(ymin,array(i))
```

```
10        ymax=amax1(ymax,array(i))
        delta=(ymax-ymin)/axlen
        if (delta .eq. 0.0)goto 20
        array(n+1)=ymin
        array(n+1+k)=delta
        return
20      if(ymin)40,50,30
30      array(n+1)=0.0
        array(n+1+k)=(2.*ymin)/axlen
        return
40      array(n+1)=2.*ymin
        array(n+1+k)=(-2.*ymin)/axlen
        return
50      array(n+1)=-1.
        array(n+1+k)=2./axlen
        return
        end
c
c*******************************************************************************
c
        subroutine showline(ix1,iy1,ix2,iy2)
c
c       draws a line from x1,y1 to x2,y2
c
        integer*2 dx,dy,xinc,yinc,tv,a,b,tmp,x1,x2,y1,y2
        integer*2 rax,rcx,rdx,iy1,iy2
        logical*2 sflag
        integer*2 idevice,ixmaxs,iymaxs,irotation
        integer*2 ixvmin,ixvmax,iyvmin,iyvmax
        integer*2 ipenplotter,maxpenplotter,idashlinetype
        real xrmin,xrmax,yrmin,yrmax
        real xcurrent,ycurrent,rfact
        real xvmin,xvmax,yvmin,yvmax
        real xiscale,xascale,yiscale,yascale
        common /diplib/idevice
     $  ,xrmin,xrmax,yrmin,yrmax
     $  ,ixmaxs,iymaxs
     $  ,xcurrent,ycurrent
     $  ,rfact
     $  ,ixvmin,ixvmax,iyvmin,iyvmax
     $  , xvmin, xvmax, yvmin, yvmax
     $  ,irotation
     $  ,ipenplotter,maxpenplotter
     $  ,xiscale,xascale,yiscale,yascale
     $  ,aspectsymbol,talicsymbol
     $  ,idashlinetype,dashlinelenght
     $  ,ipfile,corx,cory
        if(ipfile .ne. 0)then
          write(99)ix1,iy1,ix2,iy2,ipenplotter
          x1=nint(real(       ix1)*corx)
          x2=nint(real(       ix2)*corx)
          y1=nint(real(iymaxs-iy1)*cory)
          y2=nint(real(iymaxs-iy2)*cory)
         else
          x1=ix1
          x2=ix2
          y1=iymaxs-iy1
          y2=iymaxs-iy2
          endif
        rax=3072+ipenplotter
        if(x2 .ge. x1)then
          dx=x2-x1
          xinc=1
         else
          dx=x1-x2
          xinc=-1
          endif
        if(y2 .ge. y1)then
          dy=y2-y1
          yinc=1
         else
          dy=y1-y2
          yinc=-1
          endif
        sflag=.false.
        if(dy .gt. dx)then
          sflag=.true.
          tmp=dy
          dy=dx
          dx=tmp
          endif
        rcx=x1
        rdx=y1
        tv=dy*2-dx
        a=dy*2
```

```
        b=(dy-dx)*2
        dx=dx+1
100     if(dx .eq. 0)return
        call int10(rax,rcx,rdx)
        if(tv .lt. 0)then
          tv=tv+a
         else
          tv=tv+b
          if(sflag)then
            rcx=rcx+xinc
            else
            rdx=rdx+yinc
            endif
          endif
        if(sflag)then
          rdx=rdx+yinc
         else
          rcx=rcx+xinc
          endif
        dx=dx-1
        goto 100
        end
c*******************************************************************************
        subroutine solut(x,n)
        dimension x(8,9)
        integer*2 i,in,it,j,k,l,nm1,np1
        nm1=n-1
        np1=n+1
        do 10 i=1,nm1
        l=i+1
        it=i
        do 6 in=l,n
        if(abs(x(it,i))-abs(x(in,i)))5,6,6
5       it=in
6       continue
        if(x(it,i))8,7,8
7       n=0
        return
8       if(it-i)17,17,16
16      do 9 in=i,np1
        xt=x(i,in)
        x(i,in)=x(it,in)
9       x(it,in)=xt
17      do 10 j=l,n
        ratio=x(j,i)/x(i,i)
        do 10 k=l,np1
10      x(j,k)=x(j,k)-ratio*x(i,k)
        do 40 i=1,n
        dx=0.0
        k=n-i+1
        if(i-1)40,40,20
20      do 30 j=2,i
        l=n+2-j
30      dx=dx+x(k,l)*x(l,np1)
40      x(k,np1)=(-x(k,np1)-dx)/x(k,k)
        return
        end
c*******************************************************************************
        subroutine symbol(xpage,ypage,heigh,ibcd,angle,nchar)
        integer*2 lcode(859),lstart(112),i,j,iletter,nt
        integer*2 lcode1(120),lcode2(120),lcode3(120),lcode4(120)
     $  ,lcode5(120),lcode6(120),lcode7(120),lcode8(19)
        character*1 ibcd(*)
        integer*2 idevice,ixmaxs,iymaxs,irotation
        integer*2 ixvmin,ixvmax,iyvmin,iyvmax
        integer*2 ipenplotter,maxpenplotter,idashlinetype
        real xrmin,xrmax,yrmin,yrmax
        real xcurrent,ycurrent,rfact
        real xvmin,xvmax,yvmin,yvmax
        real xiscale,xascale,yiscale,yascale
        common /diplib/idevice
     $  ,xrmin,xrmax,yrmin,yrmax
     $  ,ixmaxs,iymaxs
     $  ,xcurrent,ycurrent
     $  ,rfact
     $  ,ixvmin,ixvmax,iyvmin,iyvmax
     $  , xvmin, xvmax, yvmin, yvmax
     $  ,irotation
     $  ,ipenplotter,maxpenplotter
     $  ,xiscale,xascale,yiscale,yascale
     $  ,aspectsymbol,talicsymbol
     $  ,idashlinetype,dashlinelenght
     $  ,ipfile,corx,cory
        equivalence(lcode1,lcode(  1))
        equivalence(lcode2,lcode(121))
```

```
      equivalence(lcode3,lcode(241))
      equivalence(lcode4,lcode(361))
      equivalence(lcode5,lcode(481))
      equivalence(lcode6,lcode(601))
      equivalence(lcode7,lcode(721))
      equivalence(lcode8,lcode(841))
      data lcode1/
     $  802,578,580,516,512,576,578,802,802,578,579,564,532,515,513
     $ ,528,560,577,578,802,802,578,532,528,578,802,800,548,770,578
     $ ,802,768,580,832,516,802,802,578,548,514,544,578,802,770,578
     $ ,548,544,578,802,768,580,576,516,802,832,580,512,516,801,547
     $ ,802,770,546,576,836,546,836,546,819,531,516,787,529,512,785
     $ ,561,576,817,563,802,800,548,770,578,768,580,832,516,802,768
     $ ,580,576,516,512,802,770,578,802,802,578,532,528,578,770,560
     $ ,564,514,802,802,548,802,808,805,565,837,645,808,867,628,644/
      data lcode2/
     $  868,629,645,808,803,643,901,549,838,578,866,614,808,818,565
     $ ,582,597,595,610,627,630,900,548,808,802,562,630,646,900,642
     $ ,610,612,644,836,582,550,548,580,808,838,548,547,562,578,612
     $ ,628,643,626,610,550,808,868,629,645,808,805,579,611,645,808
     $ ,803,581,613,643,808,835,613,867,581,851,597,868,580,808,850
     $ ,598,820,628,808,787,548,564,808,850,598,808,803,563,564,548
     $ ,547,808,802,562,630,646,808,818,626,643,644,629,565,548,547
     $ ,562,808,803,549,883,644,548,808,882,643,645,630,614,597,596/
      data lcode3/
     $  562,546,550,808,898,646,595,597,582,566,549,546,808,838,578
     $ ,594,645,549,808,818,547,549,566,582,597,594,642,646,808,850
     $ ,597,582,566,549,547,562,626,643,646,808,898,646,614,546,808
     $ ,851,597,582,566,549,547,562,578,595,610,626,643,645,630,614
     $ ,597,808,802,549,566,630,645,643,626,610,595,598,808,803,563
     $ ,564,548,547,835,595,596,580,579,808,788,549,565,837,597,808
     $ ,805,594,645,808,834,582,866,614,808,803,598,643,808,882,643
     $ ,645,630,596,580,820,548,808,837,613,611,579,581,598,630,645/
      data lcode4/
     $  643,626,578,548,550,808,802,626,643,645,630,550,850,598,808
     $ ,802,549,566,582,597,595,898,645,630,614,597,899,547,808,822
     $ ,549,547,562,626,643,645,630,808,802,549,566,630,645,642,899
     $ ,547,808,902,642,546,550,853,594,808,902,642,546,853,594,808
     $ ,852,598,566,549,547,562,626,643,646,808,802,642,850,598,902
     $ ,550,808,899,645,900,548,803,549,808,818,547,548,565,645,900
     $ ,646,808,802,642,902,578,851,550,808,898,546,550,808,802,642
     $ ,612,646,550,808,802,642,582,902,550,808,818,626,643,645,630/
      data lcode5/
     $  566,549,547,562,808,803,643,898,645,630,614,597,595,808,818
     $ ,626,643,645,630,566,549,547,562,836,550,808,802,642,645,630
     $ ,614,597,594,851,550,808,818,547,549,566,582,597,595,610,626
     $ ,643,645,630,808,804,644,898,646,808,898,562,547,549,566,646
     $ ,808,898,578,548,582,646,808,898,546,580,550,646,808,802,562
     $ ,630,646,898,626,566,550,808,898,626,596,630,646,852,548,808
     $ ,898,646,630,562,546,550,808,900,643,547,548,808,898,626,566
     $ ,550,808,900,645,549,548,808,866,644,614,808,786,534,808,899/
      data lcode6/
     $  628,808,850,611,613,598,550,822,549,547,562,579,582,808,802
     $ ,642,850,611,613,598,566,549,547,562,808,854,613,611,594,562
     $ ,547,549,566,808,854,613,611,594,562,547,549,566,806,646,808
     $ ,834,582,598,613,611,594,562,547,550,808,803,627,644,645,630
     $ ,868,610,808,854,613,611,594,562,547,549,566,870,534,517,515
     $ ,530,808,802,642,850,611,613,598,550,808,804,596,868,628,808
     $ ,786,515,517,534,614,613,808,898,546,614,836,550,808,899,644
     $ ,548,803,549,808,802,610,850,611,596,548,852,613,598,550,808/
      data lcode7/
     $  802,610,850,611,613,598,550,808,854,613,611,594,562,547,549
     $ ,566,598,808,818,547,549,566,598,613,611,594,866,514,808,854
     $ ,613,611,594,562,547,549,566,870,518,808,802,610,850,611,613
     $ ,598,808,818,547,549,566,581,579,594,611,613,598,808,822,549
     $ ,548,563,643,866,612,808,866,562,547,549,566,806,614,808,866
     $ ,578,548,582,614,808,866,562,547,564,596,820,549,566,614,808
     $ ,802,614,866,550,808,866,562,547,549,566,870,534,517,515,530
     $ ,808,866,614,546,550,808,805,547,579,594,611,643,645,808,804/
      data lcode8/
     $  580,868,644,808,803,549,581,598,613,645,643,808,834,594,611
     $ ,581,598,614,808/
c
c...    lstart geeft aan op welk array element van lcode
c...    de letter begint. voor index zie calcomp manual pagina 2-5
c
      data lstart/
     $    1,  9, 21, 27, 32, 37, 44, 50, 55, 62, 67, 81, 90, 96, 99
     $ ,109,111,113,118,125,134,145,160,172,176,181,186,195,200,204
     $ ,207,213,218,228,234,245,254,260,270,281,286,303,314,325,331
     $ ,335,340,344,353,367,376,390,399,408,415,421,431,438,445,453
     $ ,460,464,470,476,486,495,507,517,530,535,542,548,554,563,571
     $ ,578,583,588,593,597,600,603,615,626,635,646,656,664,678,686
     $ ,691,698,704,710,721,729,739,750,761,768,779,787,795,801,811
     $ ,816,827,832,840,845,853,860/
```

```
       height=heigh/6.
c geen tekst
       if(abs(height).lt. 0.001)return
       xo=xpage
       yo=ypage
       nt=mod(nchar,1000)
c special mode centered symbol ************************************
       if(nchar .lt. 0)then
         iletter=ichar(ibcd(1))+1
         if(nchar .eq. -1)then
           xcurrent=xo
           ycurrent=yo
          else
           call plot(xo,yo,2)
           endif
         if(iletter .le. 16 .and. iletter .ge. 1)then
c vectoren aflopen
           do 500 j=lstart(iletter),lstart(iletter+1)-1
c            xn=xo+height*real(iand(     lcode(j)    ,15)-2)
             xn=xo+height*real(mod(lcode(j)   ,16)-2)
c            yn=yo+height*real(iand(ishl(lcode(j),-4),15)-2)
             yn=yo+height*real(mod(lcode(j)/16,16)-2)
c pen up/down
c            if(iand(ishl(lcode(j),-8),3) .eq.2)then
             if(mod(lcode(j)/256,4) .eq.2)then
               call plot(xn,yn,2)
              else
               xcurrent=xn
               ycurrent=yn
               endif
500          continue
           endif
         return
         endif
c set mode ********************************************************
       if(height .lt. -0.001)then
         if(xo .ne. 999.)aspectsymbol=xo
         if(yo .ne. 999.)then
           if(yo .ne. 0.)then
             talicsymbol=0.27
            else
             talicsymbol=0.0
             endif
           endif
         return
         endif
       th=angle*0.017453
       sinth=sin(th)
       costh=cos(th)
c... text *********************************************************
c... als x of y=999.0 dan plotten van af de current locatie
       if(xo .eq. 999.)xo=xcurrent
       if(yo .eq. 999.)yo=ycurrent
       do 200 i=1,nt
         iletter=ichar(ibcd(i))-15
c... alleen ascii
         if(iletter .lt. 17 .or. iletter .gt. 111)iletter=17
c... vectoren aflopen
         do 100 j=lstart(iletter),lstart(iletter+1)-1
c          y=real(iand(ishl(lcode(j),-4),15)-2)
           y=real(mod(lcode(j)/16,16)-2)
c          x=height*aspectsymbol*(
c    $ real(6*(i-1))+y*talicsymbol+real(iand(lcode(j),15)-2))
           x=height*aspectsymbol*(
     $ real(6*(i-1))+y*talicsymbol+real(mod(lcode(j),16)-2))
           y=height*y
c... coordinaten stelsel draaien
           xn=xo+x*costh-y*sinth
           yn=yo+x*sinth+y*costh
c... pen up/down
c          if(iand(ishl(lcode(j),-8),3) .eq. 2)then
           if(mod(lcode(j)/256,4).eq.2)then
             call plot(xn,yn,2)
            else
             xcurrent=xn
             ycurrent=yn
             endif
100        continue
200      continue
       return
       end
c*********************************************************************
       subroutine viewport(ilun,x1,x2,y1,y2,irot)
c...   set plotting viewport
c...   ilun    = logical unit number  < 0 : only rotation
```

```
c...    x1      = left  x       0.0 <= x1 <= 1.0
c...    x2      = right x       0.0 <= x2 <= 1.0
c...    y1      = left  y       0.0 <= y1 <= 1.0
c...    y2      = right y       0.0 <= y2 <= 1.0
c...    irot    = number of rotations of 90 degrees before mapping window
c...              on viewport
        integer*2 idevice,ixmaxs,iymaxs,irotation
        integer*2 ixvmin,ixvmax,iyvmin,iyvmax
        integer*2 ipenplotter,maxpenplotter,idashlinetype
        real xrmin,xrmax,yrmin,yrmax
        real xcurrent,ycurrent,rfact
        real xvmin,xvmax,yvmin,yvmax
        real xiscale,xascale,yiscale,yascale
        common /diplib/idevice
     $  ,xrmin,xrmax,yrmin,yrmax
     $  ,ixmaxs,iymaxs
     $  ,xcurrent,ycurrent
     $  ,rfact
     $  ,ixvmin,ixvmax,iyvmin,iyvmax
     $  , xvmin, xvmax, yvmin, yvmax
     $  ,irotation
     $  ,ipenplotter,maxpenplotter
     $  ,xiscale,xascale,yiscale,yascale
     $  ,aspectsymbol,talicsymbol
     $  ,idashlinetype,dashlinelenght
     $  ,ipfile,corx,cory
        if(ilun .ge. 0)then
          ixvmin=max0(min0(nint(x1*ixmaxs),ixmaxs),0)
          ixvmax=max0(min0(nint(x2*ixmaxs),ixmaxs),0)
          iyvmin=max0(min0(nint(y1*iymaxs),iymaxs),0)
          iyvmax=max0(min0(nint(y2*iymaxs),iymaxs),0)
          endif
        irotation=mod(irot,4)
        call drawscal
        return
        end
c*************************************************************************
        subroutine window(x1,x2,y1,y2)
c
c...    set plotting window
c...    x1      = left  x
c...    x2      = right x
c...    y1      = left  y
c...    y2      = right y
c
        integer*2 idevice,ixmaxs,iymaxs,irotation
        integer*2 ixvmin,ixvmax,iyvmin,iyvmax
        integer*2 ipenplotter,maxpenplotter,idashlinetype
        real xrmin,xrmax,yrmin,yrmax
        real xcurrent,ycurrent,rfact
        real xvmin,xvmax,yvmin,yvmax
        real xiscale,xascale,yiscale,yascale
        common /diplib/idevice
     $  ,xrmin,xrmax,yrmin,yrmax
     $  ,ixmaxs,iymaxs
     $  ,xcurrent,ycurrent
     $  ,rfact
     $  ,ixvmin,ixvmax,iyvmin,iyvmax
     $  , xvmin, xvmax, yvmin, yvmax
     $  ,irotation
     $  ,ipenplotter,maxpenplotter
     $  ,xiscale,xascale,yiscale,yascale
     $  ,aspectsymbol,talicsymbol
     $  ,idashlinetype,dashlinelenght
     $  ,ipfile,corx,cory
        xrmin=x1
        xrmax=x2
        yrmin=y1
        yrmax=y2
        call drawscal
        return
        end
c*************************************************************************
        subroutine where(xpage,ypage,fact)
c...    read current pen position
c...    xpage   = current x location
c...    ypage   = current y location
c...    fact    = scale factor of window
c
        integer*2 idevice,ixmaxs,iymaxs,irotation
        integer*2 ixvmin,ixvmax,iyvmin,iyvmax
        integer*2 ipenplotter,maxpenplotter,idashlinetype
        real xrmin,xrmax,yrmin,yrmax
        real xcurrent,ycurrent,rfact
        real xvmin,xvmax,yvmin,yvmax
```

```
      real xiscale,xascale,yiscale,yascale
      common /diplib/idevice
     $  ,xrmin,xrmax,yrmin,yrmax
     $  ,ixmaxs,iymaxs
     $  ,xcurrent,ycurrent
     $  ,rfact
     $  ,ixvmin,ixvmax,iyvmin,iyvmax
     $  , xvmin, xvmax, yvmin, yvmax
     $  ,irotation
     $  ,ipenplotter,maxpenplotter
     $  ,xiscale,xascale,yiscale,yascale
     $  ,aspectsymbol,talicsymbol
     $  ,idashlinetype,dashlinelenght
     $  ,ipfile,corx,cory
      xpage=xcurrent
      ypage=ycurrent
      fact=rfact
      return
      end
```

```
        PAGE    ,80
        title   PEEK routine
;               aanroep Peek(
;                       segment adres,
;                       offset adres,
;                       waarde)
;
;       parameter list control block
frame   struc
savebp  dw      ?       ;save copy of BP register
saveds  dw      ?       ;save copy of DS register
retaddr dd      ?       ;return address
datab   dd      ?       ;data
offads  dd      ?       ;offset adres
segads  dd      ?       ;segment adres
frame   ends

data    segment public 'data'
data    ends

dgroup  group   data
myseg   segment 'code'
        assume cs:myseg,ds:dgroup,ss:dgroup
public  peek
peek    proc    far
        push    bp
        push    ds
        mov     bp      ,sp
        les     bx      ,[bp].offads
        mov     cx      ,es:[bx]
        les     bx      ,[bp].segads
        mov     dx      ,es
        mov     es      ,es:[bx]        ;segment adres
        mov     bx      ,cx             ;update offset
        mov     al      ,es:[bx]
        mov     ah      ,0              ;alleen byte waarde overdragen
        mov     es      ,dx
        les     bx      ,[bp].datab
        mov     es:[bx] ,ax
        pop     ds
        pop     bp
        ret     12                      ;return to fortran
peek    endp
myseg   ends
        end
```

```
C       SCIENTIFIC SUBROUTINES PACKAGE
C...SUBROUTINE CURVX
C... (XO, XF, COEFF1, EXP1, COEFF2, EXP2, COEFF3, EXP3, COEFF4, EXP4)
C...
C...XO, XF                              ARE THE STARTING AND ENDING
C...                                    VALUES OF X IN INCHES.
C...COEFF1, COEFF2, COEFF3, COEFF4      ARE THE COEFFICIENTS OF THE
C...                                    POLYNOMIAL.
C...EXP1, EXP2, EXP3, EXP4              ARE THE EXPONENTS OF THE
C...                                    POLYNOMIAL.
        SUBROUTINE CURVX(X0,XF,A,E,B,F,C,G,D,H)
C...INITIALIZATION
C...GET LINE LENGTH
        DX=XF-X0
        I3=3
C...DEVELOP FACTORED DELTA
        CALL WHERE(X,Y,XFCT)
        X=X0
        DLT=0.03/XFCT
C...CHECK LINE LENGTH (IF ZERO RETURN)
        IF(DX)10,20,15
C...IF NEGATIVE MAKE DELTA LIKEWISE
10      DLT=-DLT
C...COMPUTE NUMBER OF LINE POINTS
15      N=ABS(DX/DLT)+1.0
C...CURVE FITTING PLOT LOOP
        DO 17 IV=1,N
        Y=A*X**E+B*X**F+C*X**G+D*X**H
        CALL PLOT(X,Y,I3)
        X=X+DLT
17      I3=2
C...PLOT EXPLICIT FINAL POINT AND RETURN
        Y=A*XF**E+B*XF**F+C*XF**G+D*XF**H
        CALL PLOT(XF,Y,2)
20      RETURN
        END
C...CALL CURVY
C... (YO, YF, COEFF1, EXP1, COEFF2, EXP2, COEFF3, EXP3, COEFF4, EXP4)
C...
C...YO, YF                              ARE THE STARTING AND ENDING
C...                                    VALUES OF Y IN INCHES.
C...COEFF1, COEFF2, COEFF3, COEFF4      ARE THE COEFFICIENTS OF THE
C...                                    POLYNOMIAL.
C...EXP1, EXP2, EXP3, EXP4              ARE THE EXPONENTS OF THE
C...                                    POLYNOMIAL.
        SUBROUTINE CURVY(Y0,YF,A,E,B,F,C,G,D,H)
C...INITIALIZATION -
C...GET LINE LENGTH
        DY=YF-Y0
        I3=3
C...DEVELOP FACTORED DELTA
        CALL WHERE(X,Y,XFCT)
        Y=Y0
        DLT=0.03/XFCT
C...CHECK LINE LENGTH (IF ZERO RETURN)
        IF(DY)10,20,15
C...IF NEGATIVE MAKE DELTA LIKEWISE
10      DLT=-DLT
C...COMPUTE NUMBER OF LINE POINTS
15      N=ABS(DY/DLT)+1.0
C...CURVE FITTING PLOT LOOP
        DO 17 IV=1,N
        X=A*Y**E+B*Y**F+C*Y**G+D*Y**H
        CALL PLOT(X,Y,I3)
        Y=Y+DLT
17      I3=2
C...PLOT EXPLICIT FINAL POINT AND RETURN
        X=A*YF**E+B*YF**F+C*YF**G+D*YF**H
        CALL PLOT(X,YF,2)
20      RETURN
        END
        SUBROUTINE FIT4(PX1,PY1,PX2,PY2,VECX1,VECY1,VECX3,VECY3)
        DATA VX2,VY2,VX3,VY3 /0.,0.,0.,0./, D3,UX2,UY2/0.,0.,0./
        X1=PX1
        Y1=PY1
        CALL WHERE(X,Y,D)
        D=0.01/D
        IF(ABS(X1-X)-D)10,2,2
10      IF(ABS(Y1-Y)-D)11,2,2
11      IF(VECX1-VX2)5,12,5
12      IF(VECY1-VY2)5,13,5
13      IF(VX3-PX2+X1)5,14,5
14      IF(VY3-PY2+Y1)5,6,5
2       CALL PLOT(PX1,PY1,3)
5       VX3=PX2-X1
```

```
      VY3=PY2-Y1
      VX2=VECX1
      VY2=VECY1
      D2=VX2*VX2+VY2*VY2
      T=1.0
      GOTO 7
6     T=0.0
      VX2=VX3
      VY2=VY3
      VX3=VECX3
      VY3=VECY3
      D2=D3
      UX1=UX2
      UY1=UY2
7     D3=VX3*VX3+VY3*VY3
      UX2=D2*VX3+D3*VX2
      UY2=D2*VY3+D3*VY2
      DV=1.0/SQRT(UX2*UX2+UY2*UY2+0.00001)
      UX2=DV*UX2
      UY2=DV*UY2
      IF(T)6,8,6
8     D=ABS(UX1*VX2+UY1*VY2)
      D1=D
      UUX1=D*UX1
      UUY1=D*UY1
      D=ABS(UX2*VX2+UY2*VY2)
      UUX2=D*UX2
      UUY2=D*UY2
      D=D+D1
      AX=UUX2+UUX1-VX2-VX2
      BX=VX2-UUX1-AX
      AY=UUY2+UUY1-VY2-VY2
      BY=VY2-UUY1-AY
      N=10.*D+1.
      D=1./FLOAT(N)
      DO 9 I=1,N
      T=T+D
      X=((AX*T+BX)*T+UUX1)*T+X1
      Y=((AY*T+BY)*T+UUY1)*T+Y1
9     CALL PLOT(X,Y,2)
      RETURN
      END
      SUBROUTINE FLINE(X,Y,NN,K,J,L)
C     X  IS THE NAME OF THE ARRAY OF UNSCALED ORDINATE VALUES.
C     Y  IS THE NAME OF THE ARRAY OF UNSCALED ABCISSA VALUES.
C     N  IS THE NUMBER OF POINTS IN THE ARRAY, NEGATIVE TO FIT
C     K  IS THE REPEAT CYCLE OF A MIXED ARRAY (NORMALLY = 1).
C     J  IS THE ALTERNATE NUMBER OF DATA POINT TO PLOT A SYMBOL.
C           J WILL = 0 FOR LINE PLOT,NEGATIVE FOR POINT PLOT,
C           J = 1 FOR POINT FOR EVERY DATA POINT,2 FOR EVERY OTHER
C     L  IS AN INTEGER DESCRIBING SYMBOL TO BE USED, SEE SYMBOL
C           ROUTINE FOR LIST
C
C     NOTE THIS ROUTINE EXPECTS  XMIN,DX,YMIN AND DY TO BE STORED IN
C     X(N*K+1),X(N*K+1+K),Y(N*K+1),AND Y(N*K+1+K) RESPECTIVELY.
C
      DIMENSION X(2),Y(2)
      INTEQ=L
      N=IABS(NN)
      KK=K
      LMIN=N*KK+1
      LDX=LMIN+KK
      NL=LMIN-KK
      XMIN=X(LMIN)
      YMIN=Y(LMIN)
      DX=X(LDX)
      DY=Y(LDX)
      CALL WHERE(XN,YN,DF)
      DF=AMAX1(ABS((X( 1)-XMIN)/DX-XN),ABS((Y( 1)-YMIN)/DY-YN))
      DL=AMAX1(ABS((X(NL)-XMIN)/DX-XN),ABS((Y(NL)-YMIN)/DY-YN))
      IC=3
      IS=-1
      NT=IABS(J)
      IF(J)2,1,2
1     NT=1
2     IF(DF-DL)3,3,4
3     NF=1
      NA=NT
      GOTO 5
4     KK=-KK
      NF=NL
      NL=1
      NA=((N-1)/NT)*NT+NT-N+1
5     IF(J)6,7,8
6     ICA=3
```

```
      ISA=-1
      LSW=1
      GOTO 9
7     NA=LDX
8     ICA=2
      ISA=-2
      LSW=0
9     XN1=(X(NF)-XMIN)/DX
      YN1=(Y(NF)-YMIN)/DY
      NF=NF+KK
      IF(NN)10,10,25
10    X0=XN1
      Y0=YN1
      LP=NL
      NLP=LP-KK
      NFP=NF-KK
      U=(X(NF)-X(NFP))/DX
      V=(Y(NF)-Y(NFP))/DY
      U1=(X(LP)-X(NLP))/DX
      V1=(Y(LP)-Y(NLP))/DY
      SU=U
      SV=V
      IF(X(NFP)-X(LP))13,12,13
12    IF(Y(NFP)-Y(LP))13,25,13
13    NFP=NLP-KK
      SU=(X(NLP)-X(NFP))/DX
      SV=(Y(NLP)-Y(NFP))/DY
      CALL REFLX(U1,V1,SU,SV)
      NFP=NF+KK
      U1=(X(NFP)-X(NF))/DX
      V1=(Y(NFP)-Y(NF))/DY
      CALL REFLX(U,V,U1,V1)
25    DO 23 I=1,N
      XN=XN1
      YN=YN1
      IF(N-I)11,11,14
14    XN1=(X(NF)-XMIN)/DX
      YN1=(Y(NF)-YMIN)/DY
11    NW=NA-NT
      IF(NW)29,26,26
29    IF(LSW)17,26,17
26    IF(NN)16,24,15
15    CALL PLOT(XN,YN,IC)
      GOTO 20
16    IF(IC-2)15,17,15
17    IF(N-I)27,27,18
27    U2=SU
      V2=SV
      GOTO 19
18    U2=XN1-XN
      V2=YN1-YN
19    CALL FIT4(X0,Y0,XN,YN,U1,V1,U2,V2)
      U1=U
      V1=V
      U=U2
      V=V2
      X0=XN
      Y0=YN
20    NA=NA+1
      IF(NW)22,28,22
28    CALL SYMBOL(XN,YN,0.2032,CHAR(INTEQ),0.0,-1)
      NA=1
22    IC=ICA
23    NF=NF+KK
24    RETURN
      END
      SUBROUTINE LGAXS(XO,YO,IBCD,N,DIST,THETA,VORG,DELTA)
C     (XO,YO)  COORDINATES OF START OF AXIS.
C     IBCD     AXIS TITLE, EITHER HOLLERITH LITERAL OR  BCD ARRAY.
C     N        NUMBER OF CHARACTERS IN IBCD TITLE.  A NEGATIVE N PLACES
C              ANNOTATION ON CLOCK-WISE SIDE OF AXIS.
C     DIST     LENGTH OF AXIS IN PAGE INCHES.
C     THETA    ANGLE OF AXIS FROM X-DIRECTION, IN DEGREES.
C     VORG     VALUE OF DATA AT START OF AXIS.
C     DELTA    LOG-CYCLES PER INCH OR EQUIVALENTLY THE RECIPROCAL OF
C              THE LENGTH OF ONE CYCLE.
C              IF SCALOG IS USED TO SCALE A LOGARITHMIC ARRAY, VARRAY,
C              VORG=VARRAY(NV*K+1), AND DELTA=VARRAY(NV*K+K+1) WHERE
C              NV IS NUMBER OF VALUES USED AND K IS REPEAT CYCLE OF
C
C              LOCATION OF VALUES IN ARRAY, AS DESCRIBED IN SCALOG.
      DIMENSION BLOG(10)
      CHARACTER*1 IBCD(*)
C
C
```

```
C...SAVE ARGUMENTS IN X, Y, NC, SIZE.
        X=XO
        Y=YO
        NC=N
        SIZE=DIST
C...CONVERT DEGREES TO RADIANS, STORING IN TH.
        TH=0.01745329*THETA
C...STORE LOGS OF INTEGERS 2-10
        ETO10=0.4342945
        DO 10 I=1,9
10      BLOG(I)=ETO10*ALOG(FLOAT(I))
C...SET FXMN TO GREATEST INTEGER POWER OF TEN LESS THAN OR EQUAL TO LOG
C       OF XMIN.
        FXMN=INT(ETO10*ALOG(VORG)+100.0001)-100
C...CALCULATE LENGTH FROM BEGINNING OF CYCLE CONTAINING VORG TO BEGINNING
C       OF AXIS, PLUS FACTOR PREVENTING ROUND-OFF ERROR, STORING IN BLMN.
        BLMN=(ETO10*ALOG(VORG)-FXMN)/DELTA-0.0001
C...STORE SIN AND COS OF TH.
        SINT=SIN(TH)
        COST=COS(TH)
C...SET OFFSET CONSTANTS OF ANNOTATION, DEPENDING ON SIGN OF NC.
        IF(NC .LT. 0)THEN
        D1=0.6096*SINT
        D2=-0.6096*COST
        D3=0.3048*SINT-D2
        D4=-0.3048*COST+D1
        D5=0.508*SINT-0.0762*COST
        D6=-0.508*COST-0.0762*SINT
        NC=-NC
        SONT=SINT
        CIST=COST
        BCDX=X+(SIZE-0.3048*FLOAT(NC))/2.*COST+1.2192*SINT
        BCDY=Y+(SIZE-0.3048*FLOAT(NC))/2.*SINT-1.2192*COST
        ELSE IF(NC .GT. 0)THEN
        D1=-0.254*SINT
        D2=0.254*COST
        D3=-0.5588*SINT+0.6096*COST
        SONT=-SINT
        CIST=-COST
        D4=0.5588*COST+0.6096*SINT
        D5=D1-0.03*COST
        D5=D1-0.0762*COST
        D6=D2-0.0762*SINT
        BCDX=X+(SIZE-0.3048*FLOAT(NC))/2.*COST-0.8636*SINT
        BCDY=Y+(SIZE-0.3048*FLOAT(NC))/2.*SINT+0.8636*COST
        END IF
C...CALCULATE COORDINATES OF START OF CYCLE CONTAINING VORG,
C       AND STORE IN X0, Y0.
        X0=X-BLMN*COST
        Y0=Y-BLMN*SINT
C...CALCULATE LENGTH OF AXIS PLUS LENGTH OF CYCLE PRECEEDING AXIS PLUS
C       ROUND-OFF ERROR FACTOR, AND STORE IN SIZE1.
        SIZE1=SIZE+BLMN+0.0002
C...INITIALIZE CYCLE COUNTER FJ.
        FJ=0.
C...MOVE PEN TO START OF AXIS.
        CALL PLOT(X,Y,3)
C...LOOP THRU CYCLE.
C...AI DETERMINES HEIGHT OF TIC MARK, LARGE TIC MARK FOR 10**N AXIS VALUE
55      AI=0.3556
        DO 60 I=1,9
C...CALCULATE NEW BLEN, LENGTH TO NEXT TIC MARK.
        BLEN=(BLOG(I)+FJ)/DELTA
C...IF TIC MARK IS BEFORE START OF AXIS, GOTO NEXT TIC MARK.
        IF(BLEN-BLMN)60,56,56
C...IF TIC MARK IS BEYOND END OF AXIS, GOTO DRAW LINE TO END OF AXIS.
56      IF(BLEN-SIZE1)57,57,70
C...CALCULATE COORDINATES OF TIC MARK AND PLOT IT.
57      X=X0+BLEN*COST
        Y=Y0+BLEN*SINT
        CALL PLOT(X,Y,2)
        CALL PLOT(X+AI*SONT,Y-AI*CIST,2)
        CALL PLOT(X,Y,2)
60      AI=.1778
C...INCREMENT FJ TO NEXT CYCLE.
        FJ=FJ+1.
C...RETURN FOR NEXT CYCLE.
        GOTO 55
C...DRAW LINE TO END OF AXIS.
70      CALL PLOT(X0+SIZE1*COST,Y0+SIZE1*SINT,2)
C...LOOP BACKWARD THRU CYCLE FOR ANNOTATING TIC MARKS.
85      DO 110 K=1,9
        I=10-K
C...CALCULATE DISTANCE FROM START OF FIRST CYCLE TO TIC MARK.
        BLEN=(BLOG(I)+FJ)/DELTA
```

```
C...IF TIC MARK IS LOCATED BEFORE START OF AXIS, GOTO DRAW AXIS TITLE.
      IF(BLEN-BLMN)120,86,86
C...IF TIC MARK IS BEYOND END OF AXIS, GOTO NEXT TIC MARK.
86    IF(BLEN-SIZE1)87,87,110
C...IF TIC MARK IS AT INTEGER POWER OF 10, ANNOTATE WITH 10 AND EXPONENT.
87    IF(I-1)100,90,100
90    CALL NUMBER(X0+BLEN*COST+D1,Y0+BLEN*SINT+D2,0.35,10.,THETA,-1)
      CALL NUMBER(X0+BLEN*COST+D3,Y0+BLEN*SINT+D4,.175,FXMN+FJ,THETA,-1)
      GOTO 110
C...IF CYCLE LENGTH IS LESS THAN 2 INCHES, GOTO NEXT TIC MARK.
100   IF(DELTA-0.2)105,105,110
C...ANNOTATE INTERMEDIATE TIC MARK.
105   CALL NUMBER(X0+BLEN*COST+D5,Y0+BLEN*SINT+D6,
     $ .2667,FLOAT(I),THETA,-1)
110   CONTINUE
C...DECREMENT CYCLE COUNTER.
      FJ=FJ-1.
C...GOTO LOOP THRU NEXT CYCLE.
      GOTO 85
C...TEST FOR ANNOTATING AXIS TITLE.
120   IF(NC)125,130,125
C...DRAW AXIS TITLE.
125   CALL SYMBOL(BCDX,BCDY,0.3556,IBCD,THETA,NC)
130   RETURN
      END
      SUBROUTINE LGLIN(XARRA,YARRA,NV,K,JTYPE,NSY,LGTYP)
C     XARRAY  ARRAY CONTAINING VALUES TO BE PLOTTED AS THE ABSCISSAS,
C             EITHER LOGARITHMIC OR LINEAR.
C     YARRAY  ARRAY CONTAINING VALUES TO BE PLOTTED AS THE ORDINATES,
C             EITHER LOGARITHMIC OR LINEAR.
C             XARRAY AND YARRAY MUST CONTAIN SCALING FACTORS. MINIMUM
C             VALUES MUST BE LOCATED AT (NV*K+1) AND DELTA VALUES MUST
C             BE LOCATED AT (NV*K+K+1).
C     NV      NUMBER OF DATA POINTS TO BE PLOTTED.
C     K       REPEAT CYCLE OF LOCATION OF VALUES IN ARRAYS.
C     JTYPE   CONTROLS TYPE OF LINE PRODUCED.
C             JTYPE=0 PRODUCES A LINE PLOT ONLY.
C             JTYPE GREATER THAN ZERO PRODUCES A LINE PLOT WITH A
C             SYMBOL AT EVERY JTYPE-TH POINT THRU ARRAYS.
C             JTYPE LESS THAN ZERO PRODUCES ONLY SYMBOLS (NOT CONNECTED)
C             AT EVERY JTYPE-TH POINT THRU ARRAYS.
C     NSY     INTEGER SPECIFYING CENTERED SYMBOL OF SYMBOL TABLE TO
C             BE DRAWN AT EVERY JTYPE-TH DATA POINT.
C     LOGTYP  INTEGER SPECIFYING MODE OF PLOT, EITHER LOG-LOG
C             OR SEMI-LOG.
C             LOGTYP=0 PRODUCES A LOG-LOG PLOT.
C             LOGTYP=1 PRODUCES A SEMI-LOG PLOT LINEAR IN X.
C             LOGTYP=-1 PRODUCES A SEMI-LOG PLOT LINEAR IN Y.
      DIMENSION XARRA(2), YARRA(2)
      INTEQ=NSY
      ETO10=0.4342945
      LMN=NV*K+1
      LDX=LMN+K
      NL=LMN-K
C...STORE SCALING FACTORS.
      XMIN=XARRA(LMN)
      DX=XARRA(LDX)
      YMIN=YARRA(LMN)
      DY=YARRA(LDX)
C...STORE COORDINATES OF ENDS OF LINE.
      X1=XARRA(1)
      X2=XARRA(NL)
      Y1=YARRA(1)
      Y2=YARRA(NL)
C...CONVERT LINEAR TO LOG, DEPENDING ON VALUE OF LOGTYP.
      IF(LGTYP .LE. 0)THEN
      XMIN=ETO10*ALOG(XMIN)
      X1=ETO10*ALOG(X1)
      X2=ETO10*ALOG(X2)
      ENDIF
      IF(LGTYP .GE. 0)THEN
      YMIN=ETO10*ALOG(YMIN)
      Y1=ETO10*ALOG(Y1)
      Y2=ETO10*ALOG(Y2)
      ENDIF
C...LOCATE PEN.
      CALL WHERE(XN,YN,Z)
C...FIND MAXIMUM OF COORDINATES OF END POINTS OF LINE.
      DF=AMAX1(ABS((X1-XMIN)/DX-XN),ABS((Y1-YMIN)/DY-YN))
      DL=AMAX1(ABS((X2-XMIN)/DX-XN),ABS((Y2-YMIN)/DY-YN))
C...SET CONSTANTS FOR POINT PLOT, LINE PLOT, OR LINE AND SYMBOL PLOT.
C     IC  PEN UP-DOWN FOR PLOT.
C     IS  PEN UP-DOWN FOR SYMBOL.
C     NA  STEP FROM 1 TO NT.
C     NT  WHEN NA=NT, USE SYMBOL.
```

```
C       NF  SUBSCRIPT OF ARRAY VALUE TO BE PLOTTED.
C       KK  STEP NF FORWARD OR BACKWARD THRU ARRAYS.
C       ICA,ISA  VALUES OF IC AND IS AFTER FIRST POINT PLOTTED.
C       LSW  FLAG TO SKIP PLOT CALL FOR POINT PLOT ONLY.
        IC=3
        IS=-1
        NT=IABS(JTYPE)
        IF(JTYPE .EQ. 0)NT=1
        IF(DF-DL .GT. 0)THEN
        NF=NL
        NA=((NV-1)/NT)*NT+NT-(NV-1)
        KK=-K
        ELSE
        NF=1
        NA=NT
        KK=K
        ENDIF
        IF(JTYPE)100,110,120
100     ICA=3
        ISA=-1
        LSW=1
        GOTO 130
110     NA=LDX
120     ICA=2
        ISA=-2
        LSW=0
C...BEGIN DO-LOOP FOR PLOTTING.
130     DO 230 I=1,NV
C...STORE COORDINATES.
        XN=XARRA(NF)
        YN=YARRA(NF)
C...CONVERT LINEAR TO LOG DEPENDING ON VALUE OF LOGTYP.
        IF(LGTYP .LE. 0)XN=ETO10*ALOG(XN)
        IF(LGTYP .GE. 0)YN=ETO10*ALOG(YN)
C...CALCULATE PAGE COORDINATES OF POINT.
        XN=(XN-XMIN)/DX
        YN=(YN-YMIN)/DY
C...TEST FOR SYMBOL OR POSSIBLE PLOT CALL.
        IF(NA-NT)180,190,200
C...TEST FOR PLOT OR NO-PLOT.
180     IF(LSW)210,200,210
190     CALL SYMBOL(XN,YN,0.2032,CHAR(INTEQ),0.0,IS)
        NA=1
        GOTO 220
200     CALL PLOT(XN,YN,IC)
C...RESET CONSTANTS.
210     NA=NA+1
220     NF=NF+KK
        IS=ISA
230     IC=ICA
        RETURN
        END
C...SUBROUTINE POLAR(RARRAY,AARRAY,NPTS,INC,LINTYP,INTEQ,RMAX,DR)
C...  RARRAY IS THE ARRAY CONTAINING THE RADIAL VALUES OF THE POINTS
C...           TO BE PLOTTED, IN INCHES.
C...  AARRAY  IS THE ARRAY CONTAINING THE ANGULAR VALUES OF THE POINTS
C...           TO BE PLOTTED, IN RADIANS.
C...  NPTS     IS THE NUMBER OF DATA POINTS TO BE PLOTTED.
C...  INC      IS THE INCREMENT BETWEEN ELEMENTS IN THE ARRAY. INC IS
C...           GREATER THAN 1 IF THE VALUES TO BE PLOTTED ARE IN A
C...           MIXED ARRAY. NORMALLY INC=1.
C...  LINTYP  IS THE TYPE OF GRAPH TO BE PLOTTED. IF LINTYP EQUALS
C...           0       A LINE IS PLOTTED BETWEEN SUCCESSIVE DATA POINTS.
C...           1       A LINE PLOT IS PRODUCED, WITH A SYMBOL AT EACH
C...                   DATA POINT.
C...           2       A LINE PLOT IS PRODUCED, WITH A SYMBOL AT EVERY
C...                   SECOND DATA POINT.
C...           N       A LINE PLOT IS PRODUCED, WITH A SYMBOL AT EVERY
C...                   NTH DATA POINT.
C...           -N      CONNECTING LINES ARE NOT PLOTTED; A SYMBOL APPEARS
C...                   AT EVERY NTH DATA POINT.
C...  INTEQ    IS THE INTEGER EQUIVALENT OF THE SYMBOL TO BE PLOTTED
C...           AT EVERY NTH DATA POINT.
C...  RMAX     IS THE MAXIMUM RADIUS FOR THE PLOTTING AREA. IF RMAX IS
C...           POSITIVE, POLAR PERFORMS THE SCALING, AND RETURNS
C...           THE SCALE FACTOR IN DR.
C...           NEGATIVE, DR IS USED AS THE SCALE FACTOR.
C...  DR       IS THE SCALE FACTOR WHEN RMAX IS NEGATIVE.
C...           DR RETURNS THE SCALE FACTOR WHEN RMAX IS POSITIVE.
C...
        SUBROUTINE POLAR(RADAR,ANGAR,NPTS,INC,LTYP,INTEQ,RMAX,DR)
        DIMENSION RADAR(2),ANGAR(2),TEMP(4)
        K=INC
        IND1=NPTS*K+1
        IND2=IND1+ K
```

```
      NL=IND1- K
      IF(RMAX)80,80,10
10    RMAXM=0.0
      RMINM=0.0
      DO 50 I=1,NL,K
      T=RADAR(I)
      IF(T-RMAXM)30,50,20
20    RMAXM=T
      GOTO 50
30    IF(RMINM-T)50,50,40
40    RMINM=T
50    CONTINUE
      IF(ABS(RMAXM)-ABS(RMINM))60,70,70
60    RMAXM=-RMINM
70    TEMP(1)=0.0
      TEMP(2)=RMAXM
      CALL SCALE(TEMP,RMAX,2,1)
      DR=TEMP(4)
80    CALL WHERE(RN,THN,R1)
      T=RADAR(1)/DR
      TH1=ANGAR(1)
      R1=T*COS(TH1)
      TH1=T*SIN(TH1)
      DF=ABS(R1-RN)
      R1=ABS(TH1-THN)
      IF(DF-R1)90,100,100
90    DF=R1
100   T=RADAR(NL)/DR
      TH1=ANGAR(NL)
      R1=T*COS(TH1)
      TH1=T*SIN(TH1)
      DL=ABS(R1-RN)
      R1=ABS(TH1-THN)
      IF(DL-R1)110,120,120
110   DL=R1
120   IC=3
      IS=-1
      NT=IABS(LTYP)
      IF(NT)140,130,140
130   NT=1
140   IF(DF-DL)160,160,150
150   NF=NL
      NA=((NPTS-1)/NT)*NT+NT-NPTS+1
      KK=-K
      GOTO 170
160   NF=1
      NA=NT
      KK=K
170   IF(LTYP)180,190,185
180   ICA=3
      ISA=-1
      LSW=1
      GOTO 210
185   IC=2
      GOTO 200
190   NA=IND2
200   ICA=2
      ISA=-2
      LSW=0
210   DO 260 I=1,NPTS
      TH1=ANGAR(NF)
      T=RADAR(NF)/DR
      RN=T*COS(TH1)
      THN=T*SIN(TH1)
      IF(NA-NT)220,230,240
220   IF(LSW)250,240,250
230   CALL SYMBOL(RN,THN,0.2032,CHAR(INTEQ),0.0,IS)
      NA=1
      IS=ISA
      GOTO 260
240   CALL PLOT(RN,THN,IC)
      IC=ICA
250   NA=NA+1
260   NF=NF+KK
      RETURN
      END
      SUBROUTINE REFLX(VX1,VY1,VX2,VY2)
      PS=VY1*VY1
      DS=VX1*VX1
      SS=DS+PS+0.00001
      DS=DS-PS
      PS=2.0*VX1*VY1
      TEMP=(PS*VY2+VX2*DS)/SS
      VY2=(PS*VX2-VY2*DS)/SS
      VX2=TEMP
```

```
      RETURN
      END
C...  SUBROUTINE SCALG(VARRAY,DIST,NV,K)
C...  VARRAY  ARRAY OF DATA TO BE SCALED. MUST BE NV*K+K+1 IN LENGTH.
C...  DIST    THE DISTANCE OVER WHICH THE VALUES IN VARRAY ARE TO BE
C...          PLOTTED, IN PAGE INCHES.
C...  NV      NUMBER OF VALUES IN VARRAY TO BE USED.
C...  K       REPEAT CYCLE OF LOCATION OF VALUES IN VARRAY.
      SUBROUTINE SCALG(VARRA,DIST,NV,K)
      DIMENSION VARRA(2)
      INTEGER*2 I,NP
C...  XMAX,XMIN  TRIAL MINIMUM AND MAXIMUM VALUES.
C...  T  TEMP. VALUE BEING TESTED FOR MAXIMUM OR MINIMUM,
C       ALSO ROUNDING FACTOR.
C...  NP  SUBSCRIPT AND DO-LOOP LIMIT.
      ETO10=0.4342945
C...  INITIALIZE TRIAL MINIMUM AND MAXIMUM VALUES.
      XMAX=VARRA(1)
      XMIN=XMAX
C...  CALCULATE LOCATION OF LAST POINT OF ARRAY.
      NP=NV*K
C...  FIND MINIMUM AND MAXIMUM VALUES OF ARRAY.
      DO 10 I=1,NP,K
        XMIN=AMIN1(XMIN,VARRA(I))
10      XMAX=AMAX1(XMAX,VARRA(I))
C...  CALCULATE THE GREATEST 10**N LESS THAN OR EQUAL TO THE MINIMUM
C...  VALUE OF THE ARRAY AND STORE IN THE ARRAY LOCATION(NV*K+1).
      XMIN=10.0**(INT(ETO10*ALOG(XMIN)+100.0001)-100)
      NP=NP+1
      VARRA(NP)=XMIN
      NP=NP+K
C...  SET CONSTANT TO ROUND LOG OF MAXIMUM ARRAY VALUE UP.
C...  DETERMINE DELTA VALUE(NUMBER OF LOG CYCLES COVERING ARRAY VALUES
C...  DIVIDED BY THE DISTANCE OVER WHICH DATA IS TO BE PLOTTED)AND STORE
C...  IN ARRAY LOCATION(NV*K+K+1).
      XMAX=INT(ETO10*ALOG(XMAX)-100.0001)+100
      VARRA(NP)=(XMAX-ETO10*ALOG(XMIN))/DIST
      RETURN
      END
      SUBROUTINE SMOOT(XN,YN,IC)
C     THE SMOOTH ROUTINE SIMULATES THE PLOT ROUTINE WITH A NEW PLOT
C     MODE (DRAWING A SMOOTH CURVE TO THE NEW POINT).  THE SMOOTH MODE
C     IS INITIALIZED WITH THE UNITS DIGIT OF IC = 0 (FOR AN OPEN CURVE)
C     OR = 1 (FOR A CLOSED CURVE).
C             THE VALUE OF IC FOR SMOOTHING IS THE NEGATIVE OF
C     THE PEN VALUES FOR PLOTTING.  THERE IS, THEREFORE, NO RE-ORIGINING
C     WHILE SMOOTHING.  USING POSITIVE VALUES FOR IC WHILE SMOOTHING
C     WILL BE TREATED AS A CALL TO PLOT.  TO END THE CURVE AND RETURN TO
C     THE PLOT MODE LET IC BE LESS THAN -23.
      JC=IC
      KC=JC-JC/10*10
      LC=NC-IPC
      PXN=XN
      PYN=YN
      IREP=0
      IF(KC)1,4,14
1     IF(KC+1)2,5,4
2     IF(IPC)3,3,14
3     IF(JC+24)10,10,17
4     ISW=-1
      GOTO 6
5     ISW=1
6     JSW=-1
      NC=-KC/10*10
      X3=PXN
      Y3=PYN
      MC=NC+3
9     IPC=KC
      RETURN
10    IF(IPC+1)11,13,13
11    IF(ISW-1)12,15,14
12    IF(ISW+1)14,16,14
13    KC=NC+2
      IPC=1
      CALL PLOT(X3,Y3,MC)
14    CALL PLOT(PXN,PYN,KC)
      RETURN
15    IREP=2
16    IREP=IREP+1
      KC=1
17    IF(IABS(JSW)-1)14,18,14
18    X1=X2
      Y1=Y2
      X2=X3
      Y2=Y3
```

```
      X3=PXN
      Y3=PYN
      IF(IPC+1)20,19,19
19    X3=X3-X2
      VY3=Y3-Y2
      D3=VX3*VX3+VY3*VY3
      SX1=X2
      SX2=X3
      SY1=Y2
      SY2=Y3
      GOTO 40
20    IF(JSW)21,14,23
21    IF(ISW)22,14,24
22    VX2=X3-X2
      VY2=Y3-Y2
      CALL REFLX(VX3,VY3,VX2,VY2)
      D2=VX2*VX2+VY2*VY2
      GOTO 26
23    JSW=1
24    VX2=VX3
      VY2=VY3
      VX3=X3-X2
      VY3=Y3-Y2
25    D2=D3
      UX1=UX2
      UY1=UY2
26    D3=VX3*VX3+VY3*VY3
      UX2=D2*VX3+D3*VX2
      UY2=D2*VY3+D3*VY2
      DV=1.0/SQRT(UX2*UX2+UY2*UY2+0.000001)
      UX2=DV*UX2
      UY2=DV*UY2
      IF(ISW-JSW)27,27,45
27    IF(JSW)23,14,28
28    T=0.0
      CALL WHERE(X,Y,D)
      IF(ABS(X1-X)-0.01*D)29,30,30
29    IF(ABS(Y1-Y)-0.01*D)31,30,30
30    CALL PLOT(X1,Y1,MC)
31    IF(IPC+3)32,40,32
32    D=ABS(UX1*VX2+UY1*VY2)
      D1=D
      UUX1=D*UX1
      UUY1=D*UY1
      D=ABS(UX2*VX2+UY2*VY2)
      UUX2=D*UX2
      UUY2=D*UY2
      D=D+D1
      AX=UUX2+UUX1-VX2-VX2
      BX=VX2-UUX1-AX
      AY=UUY2+UUY1-VY2-VY2
      BY=VY2-UUY1-AY
      N=10.*D+1.
      D=1./FLOAT(N)
      DO 33 I=1,N
      T=T+D
      X=((AX*T+BX)*T+UUX1)*T+X1
      Y=((AY*T+BY)*T+UUY1)*T+Y1
33    CALL PLOT(X,Y,LC)
40    IF(IREP)9,9,41
41    IREP=IREP-1
      IF(ISW)43,14,42
42    PXN=SX1
      PYN=SY1
      SX1=SX2
      SY1=SY2
      SX2=SX3
      SY2=SY3
      GOTO 18
43    CALL REFLX(VX3,VY3,VX2,VY2)
      X=VX3
      Y=VY3
      VX3=VX2
      VY3=VY2
      VX2=X
      VY2=Y
      X1=X2
      Y1=Y2
      GOTO 25
45    JSW=1
      SX3=X3
      SY3=Y3
      GOTO 40
      END
```

```
      SUBROUTINE AXISB(XPAGE,YPAGE,IBCD,NCHAR,AXLEN,ANGLE,FIRST,DELTA)
C
C...  XPAGE,YPAGE  COORDINATES OF STARTING POINT OF AXIS, IN INCHES
C...  IBCD    AXIS TITLE.
C...  NCHAR   NUMBER OF CHARACTERS IN TITLE. + FOR C.C-W SIDE.
C...  AXLEN   FLOATING POINT AXIS LENGTH IN INCHES.
C...  ANGLE   ANGLE OF AXIS FROM THE X-DIRECTION, IN DEGREES.
C...  FIRST   SCALE VALUE AT FIRST TIC MARK.
C...  DELTA   CHANGE IN SCALE BETWEEN TIC MARKS ONE INCH APART
C
      CHARACTER*1 IBCD(*)
      CHARACTER*11 INH
      CHARACTER*12 INP,INM
      CHARACTER*16 INTT
      CHARACTER*20 INHT
      CHARACTER*3  ITNE
      DATA INH /'IN HUNDREDS'/
      DATA INP /'IN THOUSANDS'/
      DATA INTT/'IN TEN THOUSANDS'/
      DATA INHT/'IN HUNDRED THOUSANDS'/
      DATA INM /'IN MILLIONS'/
C... ITNE WAS IN HET ORIGINEEL NIET GEDEFINIEERD
      DATA ITNE/'ONE'/
C
      KN=NCHAR
      A=1.0
      IF(KN .LT. 0)THEN
      A=-A
      KN=-KN
      ENDIF
      EX=0.0
      ADX=ABS(DELTA)
      IF(ADX)3,7,3
3     IF(ADX-99.0)6,4,4
4     ADX=ADX/10.0
      EX=EX+1.0
      GOTO 3
5     ADX=ADX*10.0
      EX=EX-1.0
6     IF(ADX-0.01)5,7,7
7     IF(EX .GT. 0.0)EX=EX+1.0
      XVAL=FIRST*10.0**(-EX)
      ADX=DELTA*10.0**(-EX)*2.0
      STH=ANGLE*0.0174533
      CTH=COS(STH)
      STH=SIN(STH)
      DXB=-0.254
      DYB=.381*A-0.127
      XN=XPAGE+DXB*CTH-DYB*STH
      YN=YPAGE+DYB*CTH+DXB*STH
      NTIC=AXLEN+1.0
      NT=NTIC/2
      DO 20 I=1,NTIC
      IF(I-I/2*2)16,16,15
15    CALL NUMBER(XN,YN,.2667,XVAL,ANGLE,2)
      XVAL=XVAL+ADX
      XN=XN+CTH+CTH
      YN=YN+STH+STH
16    IF(NT)20,11,20
11    Z=KN
      IF(EX)12,13,12
12    Z=Z+7.0
13    DXB=-.1778*Z+AXLEN*0.5
      DYB=0.82255*A-0.1905
      XT=XPAGE+DXB*CTH-DYB*STH
      YT=YPAGE+DYB*CTH+DXB*STH
      CALL SYMBOL(XT,YT,0.3556,IBCD,ANGLE,KN)
      IF(EX)14,20,14
14    Z=KN+2
      XT=XT+Z*CTH*0.3556
      YT=YT+Z*STH*0.3556
      IF(EX)29,20,21
21    IEX=EX+.5
      IF(IEX-6)22,22,29
22    GOTO(29,23,24,25,26,27),IEX
23    CALL SYMBOL(XT,YT,0.3556,INH,ANGLE,11)
      GOTO 20
24    CALL SYMBOL(XT,YT,0.3556,INP,ANGLE,12)
      GOTO 20
25    CALL SYMBOL(XT,YT,.3556,INTT,ANGLE,16)
      GOTO 20
26    CALL SYMBOL(XT,YT,.3556,INHT,ANGLE,20)
      GOTO 20
27    CALL SYMBOL(XT,YT,0.3556,INM,ANGLE,12)
      GOTO 20
```

```
29      CALL SYMBOL(XT,YT,.3556,ITNE,ANGLE,3)
        XT=XT+(3.0*CTH-0.8*STH)*0.3556
        YT=YT+(3.0*STH+0.8*CTH)*0.3556
        CALL NUMBER(XT,YT,0.1778,EX,ANGLE,-1)
20      NT=NT-1
        CALL PLOT(XPAGE+AXLEN*CTH,YPAGE+AXLEN*STH,3)
        DXB=-0.1778*A*STH
        DYB=+0.1778*A*CTH
        A=NTIC-1
        XN=XPAGE+A*CTH
        YN=YPAGE+A*STH
        DO 30 I=1,NTIC
        CALL PLOT(XN,YN,2)
        CALL PLOT(XN+DXB,YN+DYB,2)
        CALL PLOT(XN,YN,2)
        XN=XN-CTH
30      YN=YN-STH
        RETURN
        END
C...
        SUBROUTINE AXISC(X,Y,IBCD,NC,SIZE,THETA,YMIN,DY)
        CHARACTER*3 IM(12),IMC
        CHARACTER*1 IBCD(*)
        DATA IM/'JAN','FEB','MAR','APR','MAY','JUN','JUL'
     $   ,'AUG','SEP','OCT','NOV','DEC'/
        SIG=1.0
        IF(NC .LT. 0)SIG=-1.0
        NAC=IABS(NC)
        TH=THETA * 0.017453
        N=SIZE+0.5
        CTH=COS(TH)
        STH=SIN(TH)
        TN=N
        XB=X
        YB=Y
        XA=X-.254*SIG*STH
        YA=Y+0.254*SIG*CTH
        CALL PLOT(XA,YA,3)
        DO 20 I=1,N
        CALL PLOT(XB,YB,2)
        XC=XB+CTH
        YC=YB+STH
        CALL PLOT(XC,YC,2)
        XA=XA+CTH
        YA=YA+STH
        CALL PLOT(XA,YA,2)
        XB=XC
20      YB=YC
        ADY=ABS(DY)
        ABSV=YMIN+DY * TN
        XA=XB-(.508*SIG-.127)*STH-.375*CTH
        YA=YB+(.508*SIG-.127)*CTH-.375*STH
        IDY=DY
        IY=YMIN
        IMIN=IY+(N-1)*IDY
200     IF(IMIN-12)21,21,210
210     IMIN=IMIN-12
        GOTO 200
21      DO 30 I=1,N
        XX=XA-.5*CTH
        YY=YA-.5*STH
        IMC=IM(IMIN)
        CALL SYMBOL(XX,YY,0.254,IMC,THETA,3)
        IMIN=IMIN-IDY
        IF(IMIN .LE. 0)IMIN=12+IMIN
        XA=XA-CTH
30      YA=YA-STH
C...    DO LABEL
        TNC=NAC
        XA=X+(SIZE/2.-.1778*TNC)*CTH-(-.07+SIG*.3600)*STH*2.54
        YA=Y+(SIZE/2.-.1778*TNC)*STH+(-.07+SIG*.3600)*CTH*2.54
        CALL SYMBOL(XA,YA,0.3556,IBCD,THETA,NAC)
        RETURN
        END
C...
C...    SUBROUTINE BAR(XPAGE, YPAGE, ANGLE, HEIGHT, WIDTH, SH, IHAT, NPI)
C...
C...    XPAGE, YPAGE ARE THE COORDINATES OF THE LOWER LEFT CORNER OF THE
C...            BAR, IN INCHES.
C...    ANGLE   IS THE ANGLE AT WHICH THE BASE OF THE BAR IS DRAWN,
C...            IN DEGREES. BAR IS ROTATED ABOUT (XPAGE, YPAGE).
C...    HEIGHT  IS THE HEIGHT OF THE MAIN BAR, IN INCHES.
C...    WIDTH   IS THE WIDTH OF THE BAR, IN INCHES.
C...    SH      IS THE HEIGHT OF THE INTERMEDIATE BAR, IN INCHES.
C...    IHAT    IS THE HATCHING CODE. THE INTERMEDIATE BAR IS
```

```
C...            HATCHED ACCORDING TO THIS CODE.
C...
C...            IF IHAT=1  DRAW BAR ONLY.
C...            IF IHAT=2  HATCH FROM LEFT TO RIGHT.
C...            IF IHAT=3  HATCH FROM RIGHT TO LEFT.
C...            IF IHAT=4  HATCH BOTH WAYS.
C...    NPI     IS THE NUMBER OF LINES OF HATCHING PER INCH.
C...
      SUBROUTINE BAR(X,Y,TH,H,W,SH,IHAT,NPI)
      THETA=TH*0.017453292
      XS=SIN(THETA)
      XC=COS(THETA)
      AK=W*XC
      BK=W*XS
      CALL PLOT(X,Y,3)
      X1=X-H*XS
      Y1=Y+H*XC
      CALL PLOT(X1,Y1,2)
      X1=X1+AK
      Y1=Y1+BK
      CALL PLOT(X1,Y1,2)
      X1=X+AK
      Y1=Y+BK
      CALL PLOT(X1,Y1,2)
      CALL PLOT(X,Y,2)
C     SET FLAGS ACCORDING TO IHAT
      IF(SH)9,9,7
9     RETURN
7     IF(IHAT-1)9,10,8
8     IF(IHAT-3)11,12,13
10    K1=3
14    K2=3
      GOTO 15
11    K1=2
      GOTO 14
12    K1=3
16    K2=2
      GOTO 15
13    K1=2
      GOTO 16
C     HATCH ONE
15    X2=X-SH*XS
      Y2=Y+SH*XC
      X3=X2+AK
      Y3=Y2+BK
      CALL PLOT(X3,Y3,K1)
      CALL PLOT(X2,Y2,2)
      CALL PLOT(X1,Y1,K2)
      IF(IHAT-1)9,9,25
25    FNPI=NPI
      L=W*FNPI
      M=SH*FNPI
      IF(M)9,9,701
701   IF(L)9,9,702
702   IF(L-M .LE. 0)THEN
      F=L
      ELSE
      F=M
      ENDIF
      DH=SH/F
      DW=W/F
      W2=0.0
      H1=0.0
      IH=F-1.0
      DO 50 I=1,IH
      H1=H1+DH
      H2=SH-H1
      W2=W2+DW
      W1=W-W2
      XP1=X+W1*XC
      YP1=Y+W1*XS
      CALL PLOT(XP1,YP1,3)
      CALL PLOT(X-H2*XS,Y+H2*XC,K2)
      CALL PLOT(X2+W2*XC,Y2+W2*XS,K1)
      CALL PLOT(X1-H1*XS,Y1+H1*XC,K2)
50    CALL PLOT(XP1,YP1,K1)
      GOTO 9
      END
      SUBROUTINE LBAXS(XO,YO,IBCD,N,DIST,THETA,VORG,DELTA)
C...  (XO,YO) COORDINATES OF START OF AXIS.
C...  IBCD    AXIS TITLE, EITHER HOLLERITH LITERAL OR BCD ARRAY.
C...  N       NUMBER OF CHARACTERS IN IBCD TITLE. A NEGATIVE N PLACES
C...          ANNOTATION ON CLOCK-WISE SIDE OF AXIS.
C...  DIST    LENGTH OF AXIS IN PAGE INCHES.
C...  THETA   ANGLE OF AXIS FROM X-DIRECTION, IN DEGREES.
```

```
C...    VORG    VALUE OF DATA AT START OF AXIS.
C...    DELTA   LOG-CYCLES PER INCH OR EQUIVALENTLY THE RECIPROCAL OF
C...            THE LENGTH OF ONE CYCLE.
C...            IF SCALOG IS USED TO SCALE A LOGARITHMIC ARRAY, VARRAY,
C...            VORG=VARRAY(NV*K+1), AND DELTA=VARRAY(NV*K+K+1) WHERE
C...            NV IS NUMBER OF VALUES USED AND K IS REPEAT CYCLE OF
C...            LOCATION OF VALUES IN ARRAY, AS DESCRIBED IN SCALOG.
C
        DIMENSION BLOG(10)
        CHARACTER*1 IBCD(*)
C...    IUNIT CONTAINS BCD NUMBERS AS WORDS FOR TIC MARK ANNOTATION.
C...    NCHR CONTAINS NUMBER OF CHARACTERS IN BCD NUMBERS.
C...    IEND CONTAINS NUMBER ENDING, EITHER S OR THS.
        DIMENSION NCHR(7)
        CHARACTER*9 IUNIT(7)
        CHARACTER*3 IEND(2)
C
        DATA IUNIT/'         ',' TEN     ',' HUNDRED ',' THOUSAND'
     $ ,' MILLION ',' BILLION ',' TRILLION'/
     $ ,IEND/'THS','T  '/,NCHR/1,4,8,9,8,8,9/
C
C...SAME IS USED TO CONCATENATE CALLS TO SYMBOL AND NUMBER
        SAME=999.0
C...SAVE ARGUMENTS IN X, Y, NC, SIZE.
        X=XO
        Y=YO
        NC=N
        SIZE=DIST
C...CONVERT DEGREES TO RADIANS, STORING IN TH.
        TH=0.01745329*THETA
        AUTO=SAME
        ETO10=0.4342945
C...STORE LOGS OF INTEGERS 1-9
        DO 10 I=1,9
10      BLOG(I)=ETO10*ALOG(FLOAT(I))
C...SET IXMN TO GREATEST INTEGER POWER OF TEN LESS THAN OR EQUAL TO LOG
C...OF XMIN.
        FXMN=INT(ETO10*ALOG(VORG)+100.0001)-100
C...CALCULATE LENGTH FROM BEGINNING OF CYCLE CONTAINING VORG TO BEGINNING
C...OF AXIS, PLUS FACTOR PREVENTING ROUND-OFF ERROR, STORING IN BLMN.
        BLMN=(ETO10*ALOG(VORG)-FXMN)/DELTA-0.0001
C...STORE SIN AND COS OF TH.
        SINT=SIN(TH)
        COST=COS(TH)
C...SET HT, HEIGHT OF NUMBER ANNOTATION, PROPORTIONAL TO CYCLE LENGTH.
        HT=.1143/DELTA
C...IF HT IS GREATER THAN MAXIMUM ALLOWABLE HEIGHT, RESET TO MAX HEIGHT.
        IF(HT-.2667 .GT. 0.0)HT=.2667
C...SET OFFSET CONSTANTS OF ANNOTATION, DEPENDING ON SIGN OF NC.
        IF(NC.LT.0)THEN
        D1=0.6096*SINT
        D2=-0.6096*COST
        D3=0.3048*SINT-D2
        D4=-0.3048*COST+D1
        D5=0.508*SINT-0.0762*COST
        D6=-0.508*COST-0.0762*SINT
        NC=-NC
        SONT=SINT
        CIST=COST
        D7=(.5842+HT)*SINT+.3556*COST
        D8=-(.5842+HT)*COST+.3556*SINT
        BCDX=X+(SIZE-0.3048*FLOAT(NC))/2.*COST+1.4224*SINT
        BCDY=Y+(SIZE-0.3048*FLOAT(NC))/2.*SINT-1.4224*COST
        ELSE IF(NC.GT.0)THEN
        D1=-0.254*SINT
        D2=0.254*COST
        D3=-0.5588*SINT+0.6096*COST
        SONT=-SINT
        CIST=-COST
        D4=0.5588*COST+0.6096*SINT
        D5=D1-0.0762*COST
        D6=D2-0.0762*SINT
        D7=-.5842*SINT+.3556*COST
        D8=.5842*COST+.3556*SINT
        BCDX=X+(SIZE-0.3048*FLOAT(NC))/2.*COST-1.0668*SINT
        BCDY=Y+(SIZE-0.3048*FLOAT(NC))/2.*SINT+1.0668*COST
        ENDIF
C...CALCULATE COORDINATES OF START OF CYCLE CONTAINING VORG,
C...AND STORE IN X0, Y0.
        X0=X-BLMN*COST
        Y0=Y-BLMN*SINT
C...CALCULATE LENGTH OF AXIS PLUS LENGTH OF CYCLE PRECEEDING AXIS PLUS
C...ROUND-OFF ERROR FACTOR, AND STORE IN SIZE1.
        SIZE1=SIZE+BLMN+0.0002
C...INITIALIZE CYCLE COUNTER FJ.
```

```
       FJ=0.
C...MOVE PEN TO START OF AXIS.
       CALL PLOT(X,Y,3)
C...LOOP THRU CYCLE.
55     AI=.3556
       DO 60 I=1,9
C...CALCULATE NEW BLEN, LENGTH TO NEXT TIC MARK.
       BLEN=(BLOG(I)+FJ)/DELTA
C...IF TIC MARK IS BEFORE START OF AXIS, GO TO NEXT TIC MARK.
       IF(BLEN-BLMN)60,56,56
C...IF TIC MARK IS BEYOND END OF AXIS, GO TO DRAW LINE TO END OF AXIS.
56     IF(BLEN-SIZE1)57,57,70
C...CALCULATE COORDINATES OF TIC MARK AND PLOT IT.
57     X=X0+BLEN*COST
       Y=Y0+BLEN*SINT
       CALL PLOT(X,Y,2)
C...AI DETERMINES HEIGHT OF TIC MARK, LARGE TIC MARK FOR 10**N AXIS VALUE
       CALL PLOT(X+AI*SONT,Y-AI*CIST,2)
       CALL PLOT(X,Y,2)
60     AI=0.1778
C...INCREMENT FJ TO NEXT CYCLE.
       FJ=FJ+1.0
       GOTO 55
C...DRAW LINE TO END OF AXIS.
70     CALL PLOT(X0+SIZE1*COST,Y0+SIZE1*SINT,2)
C...LOOP BACKWARD THRU CYCLE FOR ANNOTATING TIC MARKS.
85     DO 110 K=1,9
       I=10-K
C...CALCULATE DISTANCE FROM START OF FIRST CYCLE TO TIC MARK.
       BLEN=(BLOG(I)+FJ)/DELTA
C...IF TIC MARK IS LOCATED BEFORE START OF AXIS, GO TO DRAW AXIS TITLE.
       IF(BLEN-BLMN)120,86,86
C...IF TIC MARK IS BEYOND END OF AXIS, GO TO NEXT TIC MARK.
86     IF(BLEN-SIZE1)87,87,110
C...IF TIC MARK IS AT INTERMEDIATE VALUE, GO TO DRAW DIGIT VALUE.
87     IF(I-1)100,90,100
C...DRAW DIGIT 1 BY INTEGER-POWER-OF-TEN TIC MARK.
90     CALL NUMBER(X0+BLEN*COST+D1,Y0+BLEN*SINT+D2,.3556,1.,THETA,-1)
C...COMPUTE POWER OF TEN FOR THIS CYCLE.
       IE=FXMN+FJ
C...INITIALIZE INDEX OF ANNOTATION ENDING TO PRODUCE S, E.G. TENS.
       I3=2
C...TEST FOR REQUIRED ENDING.  IF IE=0 GO TO NEXT TIC MARK.
       IF(IE)95,110,96
C...SET ENDING INDEX TO PRODUCE THS, E.G. TENTHS.
95     I3=1
       IE=IABS(IE)
C...IF IE IS OUTSIDE RANGE FOR WORD ANNOTATION, USE DIGIT ANNOTATION.
96     IF(IE-14)97,97,190
C...IF ANNOTATION EXTENDS BEYOND END OF AXIS, GO TO NEXT TIC MARK.
97     IF(BLEN+HT*20.-SIZE1)98,98,110
C...SET INDEX OF START OF ANNOTATION, BLANK, TEN, OR HUNDRED.
98     I1=IE-(IE/3)*3+1
C...SET INDEX OF CENTER PART OF ANNOTATION, BLANK,THOUSAND,MILLION,ETC.
       I2=IE/3+3
C...DRAW PARTS OF ANNOTATION, USING AUTO-POSITIONING FEATURE OF SYMBOL.
       CALL SYMBOL(X0+BLEN*COST+D7,Y0+BLEN*SINT+D8,HT,IUNIT(I1),THETA,
     $ NCHR(I1))
       IF(I2-3)205,210,205
205    CALL SYMBOL(AUTO,AUTO,HT,IUNIT(I2),THETA,NCHR(I2))
210    CALL SYMBOL(AUTO,AUTO,HT,IEND(I3),THETA,3)
       GOTO 110
C...DRAW DIGIT ANNOTATION OF INTEGER-POWER-OF-TEN TIC MARK.
190    CALL NUMBER(AUTO,AUTO,0.3556,0.,THETA,-1)
       CALL NUMBER(X0+BLEN*COST+D3,Y0+BLEN*SINT+D4,.1778,FXMN+FJ
     $   ,THETA,-1)
       GOTO 110
C...IF CYCLE LENGTH IS LESS THAN 5 CM, GO TO NEXT TIC MARK.
100    IF(DELTA-0.2)105,105,110
C...ANNOTATE INTERMEDIATE TIC MARK.
105    CALL NUMBER(X0+BLEN*COST+D5,Y0+BLEN*SINT+D6,
     $ .2667,FLOAT(I),THETA,-1)
110    CONTINUE
C...DECREMENT CYCLE COUNTER.
       FJ=FJ-1.0
C...GO TO LOOP THRU NEXT CYCLE.
       GOTO 85
C...TEST FOR ANNOTATING AXIS TITLE.
120    IF(NC .NE. 0)THEN
C...DRAW AXIS TITLE.
       CALL SYMBOL(BCDX,BCDY,0.3556,IBCD,THETA,NC)
       ENDIF
       RETURN
       END
C...SUBROUTINE     SHADE
```

```
C...    THE SHADE SUBROUTINE MAY BE USED TO SHADE A POLYGON FORMED BY
C...    TWO LINES DEFINED BY TWO SETS OF POINTS. ANY NUMBER OF POINTS
C...    MAY BE USED.
C...
C...    CALL SHADE (XARAY1, YARAY1, XARAY2, YARAY2, DLIN, ANGLE,
C...           NPTS1, INC1, NPTS2, INC2)
C...
C...      XARAY1,YARAY1  ARE THE NAMES OF THE ARRAYS CONTAINING THE X AND
C...           Y COORDINATES OF THE DATA POINTS TO BE PLOTTED
C...           FOR LINE 1.
C...      XARAY2,YARAY2  ARE THE NAMES OF THE ARRAYS CONTAINING THE X AND
C...           Y COORDINATES OF THE DATA POINTS TO BE PLOTTED
C...           FOR LINE 2.
C...      DLIN  IS THE DISTANCE BETWEEN LINES OF SHADING.
C...      ANGLE IS THE ANGLE OF INCLINATION OF LINES OF SHADING,
C...           IN DEGREES.
C...      NPTS1 IS THE NUMBER OF DATA POINTS FORMING LINE 1.
C...      INC1  IS THE INCREMENT BETWEEN ELEMENTS OF THE ARRAYS
C...           (XARAY1 AND YARAY1) FORMING LINE 1. INC1 IS
C...           GREATER THAN 1 IF THE VALUES TO BE PLOTTED ARE IN
C...           A MIXED ARRAY.
C...      NPTS2 IS THE NUMBER OF DATA POINTS FORMING LINE 2.
C...      INC2  IS THE INCREMENT BETWEEN ELEMENTS OF THE ARRAYS
C...           (XARAY2 AND YARAY2) FORMING LINE 2. INC2 IS
C...           GREATER THAN 1 IF THE VALUES TO BE PLOTTED ARE IN
C...           A MIXED OR MULTIDIMENSIONED ARRAY.
C...
      SUBROUTINE SHADE(X1,Y1,X2,Y2,D,TH,N1,J1,N2,J2)
      DIMENSION X1(*),Y1(*),X2(*),Y2(*),TS(20)
      R(C1,C2)=C1*C+C2*S
      RC(PX,PY,XM,YM,DX,DY) =R((PX-XM)/DX,(PY-YM)/DY)
      K1=J1
      K2=J2
      N3=K1*(N1-1)+1
      N4=K2*(N2-1)+1
      T1=TH*.0174533
      C=COS(T1)
      S=SIN(T1)
      M=N3+K1
      YM1=Y1(M)
      XM1=X1(M)
      M=M+K1
      DY1=Y1(M)
      DX1=X1(M)
      M=N4+K2
      YM2=Y2(M)
      XM2=X2(M)
      M=M+K2
      DY2=Y2(M)
      DX2=X2(M)
      RYMIN=RC(Y1(1),X1(1),YM1,XM1,DY1,-DX1)
      RYMAX=RYMIN
      KK=1+K1
      KJ=1+K2
      DO 2 I=KK,N3,K1
      T1=RC(Y1(I),X1(I),YM1,XM1,DY1,-DX1)
      RYMIN=AMIN1(RYMIN,T1)
      RYMAX=AMAX1(RYMAX,T1)
2     CONTINUE
      DO 9 I=1,N4,K2
      T1=RC(Y2(I),X2(I),YM2,XM2,DY2,-DX2)
      RYMIN=AMIN1(RYMIN,T1)
      RYMAX=AMAX1(RYMAX,T1)
9     CONTINUE
3     M=0
      IP=3
      JU=-1
      RXD=RC(X1(N3),Y1(N3),XM1,YM1,DX1,DY1)
      XD=RC(X2(N4),Y2(N4),XM2,YM2,DX2,DY2)
      RYD=RC(Y1(N3),X1(N3),YM1,XM1,DY1,-DX1)
      YD=RC(Y2(N4),X2(N4),YM2,XM2,DY2,-DX2)
      CALL STACK(XD,YD,RXD,RYD,RYMIN,TS,M)
10    RXD=RC(X1(1),Y1(1),XM1,YM1,DX1,DY1)
      XD=RC(X2(1),Y2(1),XM2,YM2,DX2,DY2)
      RYD=RC(Y1(1),X1(1),YM1,XM1,DY1,-DX1)
      YD=RC(Y2(1),X2(1),YM2,XM2,DY2,-DX2)
      CALL STACK(XD,YD,RXD,RYD,RYMIN,TS,M)
11    DO 5 I=KK,N3,K1
      K=I-K1
      RXD=RC(X1(I),Y1(I),XM1,YM1,DX1,DY1)
      XD=RC(X1(K),Y1(K),XM1,YM1,DX1,DY1)
      RYD=RC(Y1(I),X1(I),YM1,XM1,DY1,-DX1)
      YD=RC(Y1(K),X1(K),YM1,XM1,DY1,-DX1)
      CALL STACK(XD,YD,RXD,RYD,RYMIN,TS,M)
5     CONTINUE
```

```
      DO 8 I=KJ,N4,K2
      K=I-K2
      RXD=RC(X2(I),Y2(I),XM2,YM2,DX2,DY2)
      XD=RC(X2(K),Y2(K),XM2,YM2,DX2,DY2)
      RYD=RC(Y2(I),X2(I),YM2,XM2,DY2,-DX2)
      YD=RC(Y2(K),X2(K),YM2,XM2,DY2,-DX2)
      CALL STACK(XD,YD,RXD,RYD,RYMIN,TS,M)
8     CONTINUE
      M=M/2*2
      IF(M)12,12,20
20    IF(M1)21,13,21
21    M1=0
      GOTO 14
13    M1=M+1
14    DO 6 J=1,M
      I=IABS(M1-J)
      CALL PLOT(R(TS(I),-RYMIN),R(RYMIN,TS(I)),IP)
      IP=IP+JU
6     JU=-JU
12    RYMIN=RYMIN+D
      IF(RYMIN-RYMAX)3,22,22
22    RETURN
      END
C...
      SUBROUTINE STACK(XD,YD,RXD,RYD,RYMIN,TS,M)
      DIMENSION TS(*)
7     XD=XD-RXD
      YD=YD-RYD
      IF(YD)18,1,18
18    RYD=RYMIN-RYD
31    TEST=RYD/YD
      T=TEST*XD+RXD
      IF(ABS(TEST-.5)-.5)16,17,1
17    RYMIN=RYMIN+.0001
      RYD=RYD+.0001
      GOTO 31
16    IF(M)2,2,30
30    DO 4 J=1,M
      IF(TS(J)-T)4,19,19
19    A=TS(J)
      TS(J)=T
      T=A
4     CONTINUE
2     M=M+1
      TS(M)=T
1     RETURN
      END
```

```
      SUBROUTINE AROHD(X1,Y1,X2,Y2,A1,A2,K1)
      XS=X1
      YS=Y1
      XE=X2
      YE=Y2
      AL=A1
      TW=A2*0.5
      KK=K1
      IF(A2)99,2,3
2     TW=AL/3.0
3     AW=TW
      IF(KK)4,99,5
4     KK=-KK
      CALL WHERE(XS,YS,ZZ)
5     XA=XE-XS
      YA=YE-YS
      ZZ=SQRT(XA*XA+YA*YA)
      IF(ZZ)6,99,6
6     STH=YA/ZZ
      CTH=XA/ZZ
      KT=KK/10
      KK=KK-10*KT
      GOTO(11,12,13,14,15,12,13,99,99),KK
11    ICB=2
      ICD=2
      ICE=2
      ICF=3
      GOTO 20
12    ICB=2
      ICD=2
      ICE=2
      ICF=2
      GOTO 20
13    ICB=2
      ICD=3
      ICE=3
      ICF=2
      GOTO 20
14    ICB=3
      ICD=2
      ICE=3
      ICF=2
      GOTO 20
15    ICB=3
      ICD=3
      ICE=3
      ICF=2
20    ICA=3
      CALL PLOT(XS,YS,3)
      IF(KT-1)23,22,21
21    CALL PLOT(XS+AL*CTH,YS+AL*STH,3)
      GOTO 24
22    CALL PLOT(XS,YS,3)
24    ICA=2
23    XA=XE-AL*CTH
      YA=YE-AL*STH
30    CALL PLOT(XA,YA,ICA)
      CALL PLOT(XA+AW*STH,YA-AW*CTH,ICB)
      CALL PLOT(XE,YE,2)
      CALL PLOT(XA-AW*STH,YA+AW*CTH,ICD)
      CALL PLOT(XA,YA,ICE)
      CALL PLOT(XE,YE,ICF)
      IF(KK-5)40,40,35
35    AW=AW-0.05
      IF(AW)37,37,36
36    CALL PLOT(XA+AW*STH,YA-AW*CTH,3)
      CALL PLOT(XE,YE,2)
      CALL PLOT(XA-AW*STH,YA+AW*CTH,ICD)
      GOTO 35
37    CALL PLOT(XE,YE,3)
40    IF(KT-2)99,41,99
41    KT=0
      ZZ=XE
      XE=XS
      XS=ZZ
      ZZ=YE
      YE=YS
      YS=ZZ
      STH=-STH
      CTH=-CTH
      AW=TW
      ICA=3
      GOTO 23
99    RETURN
      END
```

```
C
      SUBROUTINE ARROW(X,Y,N,K,NX)
      DIMENSION X(*),Y(*)
      NP=(N-1)*K+1
      N0=NP-K
      N1=NP+K
      N2=N1+K
      XMIN=X(N1)
      DX1=X(N2)
      YMIN=Y(N1)
      DY1=Y(N2)
      THETA=1.5708
      DTH=0.0
      DX=X(NP)-X(N0)
      DY=Y(NP)-Y(N0)
      IF(DY)8,9,9
8     THETA=-1.5708
9     IF(DX)10,12,11
10    DTH=3.1416
11    THETA=ATAN(DY/DX)
12    THETA=THETA+DTH
13    S30=-0.75*SIN(THETA)
      S05=S30/6.0
      C30=-0.75*COS(THETA)
      C05=C30/6.0
      X(N1)=X(NP)+(C30-S05)*DX1
      Y(N1)=Y(NP)+(C05+S30)*DY1
      N1=N1+K
      X(N1)=X(NP)+(C30+S05)*DX1
      Y(N1)=Y(NP)+(S30-S05)*DY1
      N1=N1+K
      X(N1)=X(NP)
      Y(N1)=Y(NP)
      N1=N1+K
      X(N1)=X(NP)-S30*DX1
      Y(N1)=Y(NP)+C30*DY1
      N1=N1+K
      X(N1)=X(NP)+S30*DX1
      Y(N1)=Y(NP)-C30*DY1
      NP=N+NX
      N1=NP*K+1
      X(N1)=XMIN
      Y(N1)=YMIN
      N2=N1+K
      X(N2)=DX1
      Y(N2)=DY1
      CALL LINE(X,Y,NP,K,0,0)
      NP=N*K+1
      NO=NP+K
      X(NP)=XMIN
      X(NO)=DX1
      Y(NP)=YMIN
      Y(NO)=DY1
      RETURN
      END
C
      SUBROUTINE CNTRL(X,Y,N,K)
      DIMENSION X(*),Y(*)
      IF(N-1)90,90,80
80    NP=(N-2)*K+1
      NM=NP+K+K
      ND=NM+K
      DO 4 I=1,NP,K
      J=I+K
      DY2=(Y(J)-Y(I))/11.0
      DY1=DY2*4.0
      DX2=(X(J)-X(I))/11.0
      DX1=DX2*4.0
      CALL PLOT((X(I)-X(NM))/X(ND),(Y(I)-Y(NM))/Y(ND),3)
      XT=X(I)+DX1
      YT=Y(I)+DY1
      CALL PLOT((XT-X(NM))/X(ND),(YT-Y(NM))/Y(ND),2)
      XT=XT+DX2
      YT=YT+DY2
      CALL PLOT((XT-X(NM))/X(ND),(YT-Y(NM))/Y(ND),3)
      XT=XT+DX2
      YT=YT+DY2
      CALL PLOT((XT-X(NM))/X(ND),(YT-Y(NM))/Y(ND),2)
      XT=XT+DX2
      YT=YT+DY2
      CALL PLOT((XT-X(NM))/X(ND),(YT-Y(NM))/Y(ND),3)
      CALL PLOT((X(J)-X(NM))/X(ND),(Y(J)-Y(NM))/Y(ND),2)
4     CONTINUE
90    RETURN
      END
```

```
C
      SUBROUTINE DIMEN(X0,Y0,DS,THETA,SCAL)
      DIMENSION WE(7),DE(7)
      TH=3.1416*THETA/180.0
      D=DS*SCAL
      XC=COS(TH)
      XS=SIN(TH)
      DO 10 I=1,7
      DE(I)=0.0
10    WE(I)=0.0
      WE(4)=+.75
      WE(5)=+.75
      DE(1)=-.25
      DE(2)=+.25
      DE(4)=+.125
      DE(5)=-.125
      J=0
      IF(D-99.999)406,406,404
404   FD=D/2.0-0.889
      FL=FD-0.127
      GOTO 420
406   IF(D-9.999)410,410,408
408   FD=D/2.0-0.7967
      FL=FD-0.127
      GOTO 420
410   IF(D-2.99)414,414,412
412   FD=D/2.0-0.8654
      FL=FD-0.127
      GOTO 420
414   IF(D-1.99)418,418,416
416   FD=D+1.0668
      FL=D/2.0
      J=1
      GOTO 420
418   FD=D+1.0668
      WE(4)=-WE(4)
      WE(5)=-WE(5)
      FL=-0.916
420   WE(7)=FL
      XD=X0+FD*XC+.127*XS
      YD=Y0+FD*XS-.127*XC
      I3=3
      DO 424 I=1,7
      XX=X0+WE(I)*XC-DE(I)*XS
      YY=Y0+WE(I)*XS+DE(I)*XC
      CALL PLOT(XX,YY,I3)
424   I3=2
      CALL NUMBER(XD,YD,0.254,DS,THETA,3)
      I3=3
      X1=X0+D*XC
      Y1=Y0+D*XS
      DO 428 I=1,7
      I8=8-I
      XX=X1-WE(I8)*XC-DE(I8)*XS
      YY=Y1+DE(I8)*XC-WE(I8)*XS
      CALL PLOT(XX,YY,I3)
428   I3=2
      IF(J)432,432,430
430   CALL PLOT(X1,Y1,2)
      XD=X1+XC
      YD=Y1+XS
      CALL PLOT(XD,YD,2)
432   RETURN
      END
C  SUBROUTINE LABEL(X1,Y1,X2,Y2,IBCD,NS,HEIGHT,ISIDE,DSTFLN,FPN,NN)
C  THIS ROUTINE PLOTS AN ARRAY OF SYMBOLS CENTERED ALONG  A  LINE. THE
C  SYMBOLS MUST FIT AT THE SPECIFIED HEIGHT AND LEAVE AT LEAST 0.1 INCH
C  MARGIN AT EACH END. IF NOT, THE CHARACTER HEIGHT IS MODIFIED SO THAT
C  THE SYMBOLS WILL FIT. HOWEVER, IF IT BECOMES NECESSARY TO ADJUST THE
C  CHARACTER HEIGHT TO LESS THAN 0.07 INCHES, THEN NOTHING IS PLOTTED.
C  THE SPECIFIED SYMBOLOGY MAY BE FOLLOWED BY A PLOTTED NUMBER. THE
C  NUMBER IS SPECIFIED AS A FLOATING POINT NUMBER, HOWEVER BOTH THE
C  SPECIFIED BCD SYMBOLS AND THE NUMERIC SYMBOLS REQUIRED FOR THE
C  NUMBER ARE CENTERED ALONG THE LINE. IF THE USER WANTS SPACE BETWEEN
C  THE SYMBOLS AND THE NUMBER,PROVIDE BLANKS IN THE BCD ARRAY. THIS
C  NUMBER APPENDAGING FEATURE IS PROVIDED FOR SUCH DRAFTING APPLICATIONS
C  THAT INVOLVE COMPUTED VALUES THAT FOLLOW DESCRIPTIVE INFORMATION
C       E.G.-   RADIUS=308.86
C  THIS FEATURE REQUIRES THE USE OF A CALCOMP SYMBOL ROUTINE WITH THE
C  LINE CONTINUATION FEATURE
C*********************************************************************
C       X1,Y1   - COORDINATES OF FIRST POINT -P1
C       X2,Y2   - COORDINATES OF SECOND POINT-P2
C       IBCD    - ADDRESS OF ARRAY OF SYMBOLS
C       NS      - NUMBER  OF SYMBOLS TO PLOT
```

```
C       HEIGHT  - HEIGHT OF SYMBOLS
C       ISIDE   - ABSOLUTE VALUE=1, PLOT  SYMBOLS ON CLOCKWISE SIDE OF LINE
C               - ABSOLUTE VALUE=2, PLOT  SYMBOLS ON COUNTERCW SIDE OF LINE
C               - ABSOLUTE VALUE=11, SAME AS 1,EXCEPT THAT A FLOATING
C                 POINT NUMBER IS TO BE PLOTTED ALONG WITH THE LABEL
C               - ABSOLUTE VALUE=12, SAME AS 2,EXCEPT FPN IS ALSO PLOTTED
C               - IF NEGATIVE, INVERT THE SYMBOLS 180 DEGREES WHEN THE
C                 DIRECTION OF THE LINE P1 TO P2 IS GREATER THAN 93.0
C                 DEGREES AND LESS THAN 267.0 DEGREES. THIS FEATURE FORCES
C                 SYMBOLOGY TO BE IN A READABLE ORIENTATION FOR STANDARD
C                 DRAWINGS.
C       DSTFLN  - THE DISTANCE(OR MARGIN) OF THE SYMBOLS FROM THE LINE.
C       FPN     - THE FLOATING POINT NUMBER TO BE PLOTTED WHENEVER ISIDE
C                 IS EQUAL TO 11 OR 12
C       NN      - THE NUMBER OF PLACES RIGHT OF DECIMAL POINT TO PLOT FPN
        SUBROUTINE LABEL(X1,Y1,X2,Y2,IBCD,NS,HGT,ISIDE,DST,FPN,NN)
        CHARACTER*1 IBCD(*)
C  SAME IS USED TO CONCATENATE CALLS TO SYMBOL AND NUMBER
        SAME=999.0
C  SPACING TO HEIGHT RATIO OF CHARACTERS
        SHR=1.0
C  WIDTH TO HEIGHT RATIO OF CHARACTERS
        WHR=0.57143
C  MINIMUM ANGLE FOR INVERSION
        THMIN=93.0
C  MAXIMUM ANGLE FOR INVERSION
        THMAX=267.0
C  SET THE MINIMUM CHARACTER HEIGHT ALLOWED(BY ADJUSTMENT)
        CHMIN=0.1771
C  SET THE MARGIN REQUIRED AT EACH END OF THE LINE
        ENDMA=0.254
C  SET UP ENTRY COORDINATES-THEY WILL BE REVERSED IF ISIDE IS NEGATIVE
C  AND THE LINE P1-P2 IS IN THE 2ND OR 3RD QUADRANT
        FPNT=FPN
        X11=X1
        Y11=Y1
        X22=X2
        Y22=Y2
        IS=ISIDE
C  THIS PARAMETER CONTROLS WHICH SIDE OF THE LINE THE SYMBOLS WILL BE ON
        ISA=IABS(ISIDE)
        ISA1=ISA-10
C  NSW IS A SWITCH THAT WILL CONTROL WHETHER A NUMBER IS ALSO PLOTTED
        NSW=1
        IF(ISA1)1,100,100
C   YES -A NUMBER WILL BE PLOTTED, RESET ISA TO CONTROL SIDE OF LINE
100     NSW=2
        ISA=ISA1
C  COMPUTE DELTAS
1       DX=X22-X11
        DY=Y22-Y11
C  FIND THE ANGLE OF THE LINE P1 TO P2
C  IF DX IS ZERO, DONT DIVIDE TO GET SLOPE
        IF(DX)3,2,3
C  SET TH FOR 90 DEGREES
2       TH=1.5708
C  WHEN BOTH DX AND DY ARE ZERO,EXIT(99) WITH NO PLOTTING
        IF(DY)4,99,10
C  CALCULATE SLOPE AND ANGLE(IN RADIANS) FOR A 1ST QUADRANT ANGLE
3       SLP=ABS(DY/DX)
        TH=ATAN(SLP)
        TH1=TH
        IF(DY)4,6,6
C  IT IS IN EITHER THE 3RD OR 4TH QUADRANT
4       TH=TH+3.1416
C  GOTO 10 IF IN 3RD QUADRANT
        IF(DX)10,10,5
C  IT IS IN THE 4TH QUADRANT
5       TH=6.2832-TH1
        GOTO 10
C  IT IS IN THE 1ST OR 2ND QUADRANT - GOTO 10 IF IN 1ST
6       IF(DX)7,10,10
C  IT IS IN THE 2ND QUADRANT
7       TH=3.1416-TH
C  GET ANGLE IN DEGREES
10      TH1=TH*57.295
C  DOES IT QUALIFY FOR INVERTING CHARACTERS BY 180 DEGREES
C  DOES THE ENTRY PARAMETER,ISIDE,REQUEST IT
        IF(IS)11,16,16
C  YES-
C  IS THE ANGLE GREATER THAN THMIN DEGREES
11      IF(TH1-THMIN)16,16,12
C  YES-
C  IS THE ANGLE LESS THAN THMAX DEGREES
12      IF(THMAX-TH1)16,16,13
```

```
C  YES-
C  SWITCH PARAMETERS TO INVERT THE PLOTTED SYMBOLS
C  INTERCHANGE P1 AND P2
13      X11=X2
        Y11=Y2
        X22=X1
        Y22=Y1
C  SET POSITIVE SO IT CANT INVERT AGAIN THIS ENTRY
        IS=-IS
C  SWITCH ISA(ISIDE) SO THE SYMBOLS WILL STILL BE ON THE INTENDED SIDE
C   OF THE LINE EVEN THOUGH THEY ARE INVERTED
C  THEN MAKE ANOTHER PASS THROUGH THE LOGIC TO GET THE ANGLES
        IF(ISA-1)14,14,15
14      ISA=2
        GOTO 1
15      ISA=1
        GOTO 1
C  FLOAT THE NUMBER OF CHARACTERS
C  IS A NUMBER TO BE PLOTTED
16      GOTO(167,161),NSW
C  YES- DETERMINE TOTAL NUMBER OF PLOTTED CHARACTERS INCLUDING THE NUMB.
C  INITIALIZE COUNTERS
161     I=0
        NC=1
C  IS IT NEGATIVE
        IF(FPNT)162,163,163
C  YES- ADD 1 FOR NEGATIVE SIGN
162     NC=NC+1
C  IS IT LESS THAN 10
163     IF(ABS(FPNT)-10.0)164,165,165
C  YES- SYMBOL  WILL BE A SINGLE DIGIT
164     NC=NC+1
        GOTO 166
165     I=0.4343*ALOG(ABS(FPNT))+1.0000001
C  CALCULATE TOTAL NUMBER OF CHARACTERS
166     TN=NC+I+NS+NN
        GOTO 168
167     TN=NS
C  GET THE ENTRY HEIGHT-IT MAY BE REDUCED
168     HT=HGT
        TEMP=TN*SHR-SHR+WHR
C  CALCULATE TOTAL LENGTH OF CHARACTERS
        TCW=HT*TEMP
C  CALCULATE LENGTH OF LINE
        DL=SQRT((DX**2)+(DY**2))
C  CALCULATE MARGIN ON EACH END OF LINE
        DLL=(DL-TCW)/2.0
C  THE MARGIN MUST EQUAL THE SET PARAMETER-ENDMA
        IF(DLL-ENDMA)17,19,19
C  CALCULATE A CHARACTER HEIGHT THAT WILL FIT
17      HT=(DL-2.0*ENDMA)/TEMP
C  THE CHARACTER HEIGHT MUST EXCEED A SET MINIMUM -CHMIN
        IF(HT-CHMIN)99,18,18
C  CHARACTERS EXCEED NECESSARY MINIMUM AND MARGIN IS SET TO MINIMUM
18      DLL=ENDMA
C  IF CLOCKWISE SIDE OF LINE,GOTO 20
19      IF(ISA-1)20,20,21
C  CLOCKWISE SIDE OF LINE BECAUSE  ISIDE=1
20      X=X11+DLL*COS(TH)+(HT+DST)*SIN(TH)
        Y=Y11+DLL*SIN(TH)-(HT+DST)*COS(TH)
        GOTO 22
C  COUNTER CLOCKWISE SIDE OF LINE ,BECAUSE ISIDE=2
21      X=X11+DLL*COS(TH)-DST*SIN(TH)
        Y=Y11+DLL*SIN(TH)+DST*COS(TH)
22      CALL SYMBOL(X,Y,HT,IBCD,TH1,NS)
C  IS A NUMBER TO BE PLOTTED
        GOTO(99,23),NSW
C  THIS CALL USES THE LINE CONTINUATION FEATURE OF THE SYMBOL ROUTINE
23      CALL NUMBER(SAME,SAME,HT,FPNT,TH1,NN)
99      RETURN
        END
```

```
      SUBROUTINE CRVPT(X,Y,L,NO,KO,SH,SWX,JT,MT,JX,MX,JY,MY,IN)
C     X- ARRAY OF X VALUES
C     Y- ARRAY OF Y VALUES
C     L- ARRAY OF POINT CHARACTERS
C           IF MINUS  THEN ONLY A SINGLE CHARACTER
C     N- NUMBER OF DATA POINTS
C     K- REPEAT FACTOR FOR ARRAYS
C     J- BCD ARRAY FOR T- TITLE
C           X- X-AXIS
C           Y- Y-AXIS
C     M- NUMBER OF CHARACTERS IN BCD ARRAYS
C     I- ARRAY OF PLOT TYPES
C           IF  MINUS  THEN A SINGLE INTEGER CONTAINING PLOT TYPES
C
      DIMENSION X(*),Y(*),L(*),IN(*),A(10)
      CHARACTER*80 JT,JX,JY
      CHARACTER*11 IRR
      CHARACTER*3 IZX
      CHARACTER*2 IYEQ
      CHARACTER*1 IZZ,TEKEN
      IRR ='ERROR NO!  '
      IZX ='Z=X'
      IYEQ='Y='
      IZZ ='Z'
      OPEN(UNIT=98,FILE='CRVPLT.DAT')
C...SAME IS USED TO CONCATENATE CALLS TO SYMBOL AND NUMBER
      SAME=999.0
      IM=4
      K=KO
      N=NO
      IN1=IN(1)
      IF(K)1,92,3
1     K=-K
3     NK=N*K
      IF(N)2,92,4
2     N=-N
      NK=N*K
      TH=SH-.5
      GOTO 10
4     TH=SH-1.
C...INITIALIZE, PLOT AXES AND TITLE
      SW=SWX
      IX=SW*4.+1.
      CALL SCALE2(X,SW,N,K)
      XOF=X(NK+1)
      NKK=NK+K+1
      X(NKK)=X(NKK)*2
      XDF=X(NKK)
      CALL SCALE2(Y,SH-3.0,N,K)
      YOF=Y(NK+1)
      YDF=Y(NKK)
      CALL AXIS2(0.0,0.0,JX,-MX,SW,0.0,XOF,XDF)
      CALL AXIS2(0.0,0.0,JY,MY,SH-3.0,90.0,YOF,YDF)
      CALL SYMBOL(1.27,SH-0.508,.3556,JT,0.0,MT)
C...PLOT DATA POINTS
10    H=0.0
      DO 11 I=1,NK,K
11      H=H+X(I)
      B=N
      H=H/B
      CALL SYMBOL(2.5,TH,0.25,IZX,0.,3)
      IF(H .GT. 0.0)THEN
        TEKEN='-'
       ELSE
        TEKEN='+'
        ENDIF
      WRITE(98,12)TEKEN,H
12    FORMAT(1X,'Z = X ',A1,E20.6)
      CALL FNUM(SAME,TH,.25,-H,IM)
      CALL PLOT(0.0,0.0,-3)
14    TH=TH-.5
      IF(IN1 .LT. 0)THEN
        J=(IN1/10)*10-IN1
        IN1=IN1/10
       ELSE
        J=IN1
        II=2
        ENDIF
      IF(J .EQ. 0)THEN
        MA=0
       ELSE
        MA=2
        ENDIF
      IF(L(1) .LT. 0)THEN
        MA=MA+1
```

```
          JA=-L(1)
          ENDIF
        MA=MA+1
        JB=-1
        DO 50 I=1,NK,K
        GOTO(42,44,42,44),MA
42      JA=L(I)
44      XC=(X(I)-XOF)/XDF
        YC=(Y(I)-YOF)/YDF
        IF(XC)78,45,45
45      IF(XC-SWX)47,47,78
47      IF(YC)78,48,48
48      IF(YC-SH)49,49,78
49      CALL SYMBOL(XC,YC,0.2,CHAR(JA),0.0,JB)
        GOTO(46,46,50,50),MA
46      JB=-2
50      CONTINUE
        IF(J)100,60,100
C...TEST FOR POLYNOMIAL FITS
60      IF(IN(1))62,64,64
62      J=(IN1/10)*10-IN1
        IN1=IN1/10
        GOTO 70
64      J=IN(II)
        II=II+1
70      IF(J)90,90,100
C...ERROR EXIT
78      EN=100+I
80      CALL SYMBOL(SW+3.75,.0,0.5,IRR,0.0,11)
        CALL NUMBER(SAME,0.0,0.5,EN,0.0,-1)
        CALL WHERE(XZ,YZ,ZZ)
        SW=XZ+2.5
C...EXIT
90      EX=0.0
        IF(KO)94,92,92
92      EX=SW+10.
94      CALL PLOT(EX,0.0,-3)
        CLOSE(UNIT=98)
        RETURN
C...FIND AND PLOT A CURVE
100     IF(N-J)102,102,110
102     EN=10+J
        GOTO 80
C...FIND THE COEFFICIENTS
110     CALL CRVFT(X,Y,N,K,J,A,H)
        WRITE(98,113)
113     FORMAT(1X,'Y =')
        DO 111,I=J+1,1,-1
111       WRITE(98,112)A(I),J+1-I
112     FORMAT(1X,E20.6,' * Z^',I1)
        CALL PLOT(0.0,0.0,-3)
        CALL SYMBOL(2.3,TH+0.127,.18,CHAR(J),0.,-1)
        CALL SYMBOL(3.3,TH,0.30,IYEQ,0.0,2)
C...PRINT EQUATION OF CURVE ON GRAPH
114     DO 140 JJ=1,J
        IF(A(JJ))116,140,116
116     CALL FNUM(SAME,TH,.25,A(JJ),IM)
        CALL SYMBOL(SAME,TH,0.30,IZZ,0.0,1)
124     BZ=J-JJ+1
        IF(BZ-1.0)140,140,130
130     CALL NUMBER(SAME,TH+.127,.25,BZ,0.0,-1)
140     CONTINUE
        IF(A(J+1))150,200,150
150     CALL FNUM(SAME,TH,.25,A(J+1),IM)
200     TH=TH-.5
        CALL WHERE(XQ,YQ,ZQ)
        IF(XQ-SW)202,202,201
201     SW=XQ
202     XN=J
C...PLOT ACTUAL CURVE WITH .25 CM  ACCURACY
        XN=XN*.5
        XC=0.0
        IP=3
        DO 250 IT=1,IX
        XT=XOF+XC*XDF
        YT=A(1)
        DO 220 JJ=1,J
220     YT=YT*(XT-H)+A(JJ+1)
        YC=(YT-YOF)/YDF
        IF(YC)224,230,230
224     YC=0.0
        IP=3
        GOTO 244
230     IF(YC-SH)240,240,234
234     YC=SH
```

```
        IP=3
        GOTO 244
240     IF(XC-XN)244,242,242
242     XN=XN+5.0
        CALL SYMBOL(XC,YC,0.15,CHAR(J),0.0,-2)
        GOTO  250
244     CALL PLOT(XC,YC,IP)
        IP=2
250     XC=XC+0.25
        GOTO 60
        END
C
        SUBROUTINE FNUM(X,Y,H,FL,N)
        CHARACTER*3 ITEN
        CHARACTER*4 IZERO
        CHARACTER*1 IPLS,IMIN,IPER
        ITEN ='(10'
        IZERO='+0.0'
        IPLS ='+'
        IMIN ='-'
        IPER ='.'
C
C...SAME IS USED TO CONCATENATE CALLS TO SYMBOL AND NUMBER
        SAME=999.0
        IF(FL)10,20,30
10      F=-FL
        CALL SYMBOL(X,Y,H,IMIN,0.0,1)
        GOTO 40
30      F=FL
        CALL SYMBOL(X,Y,H,IPLS,0.0,1)
40      EF=0.0
        NK=N
        K=N+1
        IF(F-10.0**K)104,102,102
102     EF=K
        F=F/10.0**K
108     IF(F-10.0)200,106,106
106     EF=EF+1.0
        F=F/10.0
        GOTO 108
104     IF(F-10.0)110,112,112
112     K=K-1
        IF(F-10.0**K)112,114,114
114     NK=NK-K
        GOTO 200
110     IF(F-0.1)116,200,200
116     IF(F-0.01)120,120,118
120     EF=EF-1.0
        F=F*10.0
        IF(F-1.0)120,200,200
118     NK=NK+1
200     CALL NUMBER(SAME,Y,H,F,0.0,NK)
        IF(EF)12,42,12
12      CALL SYMBOL(SAME,Y,H,ITEN,0.0,3)
        IF(EF-1.0)32,44,32
32      CALL NUMBER(SAME,Y+H/2.0,3.0*H/4.0,EF,0.0,-1)
44      CALL SYMBOL(SAME,Y,H,IPER,0.0,1)
42      RETURN
20      CALL SYMBOL(X,Y,H,IZERO,0.0,4)
        RETURN
        END
C
        SUBROUTINE CRVFT(X,Y,NP,K,ND,A,H)
C...CURVE FITTING ROUTINE
C       INPUT-  X,Y  ARRAYS OF POINTS
C               NP   NUMBER OF POINTS
C               K    REPEAT FACTOR
C               ND   DEGREE OF POLYBOMIAL DESIRED
C               H    AVERAGE VALUE OF X,USED FOR POLYNOMIAL
C       OUTPUT- A    ARRAY OF COEFFICIENTS FOR POLYNOMIAL
        DIMENSION X(*),Y(*),A(*),D(10,10)
        NPK=NP*K
        N=ND+1
        DO 40 I=1,N
          A(I)=0.0
          DO 20 J=1,N
20          D(I,J)=0.0
          KP=N-I
          DO 40 L=1,NPK,K
            Z=X(L)-H
            IF(Z .NE. 0)THEN
              A(I)=A(I)+Y(L)*Z**KP
              DO 30 J=1,N
                KR=2*N-I-J
30              D(I,J)=D(I,J)+Z**KR
```

```
            ENDIF
40          CONTINUE
C...CALLS SUBROUTINE TO SOLVE MATRIX EQUATION DX=A FOR X AND
C...RETURN SOLUTION IN A WITH DETERMINANT IN R
        CALL MATX(D,A,N,R)
        RETURN
        END
C
        SUBROUTINE MATX(A,X,N,R)
        DIMENSION A(10,10),X(10)
        R=1.0
        M=N-1
        DO 50 K=1,M
          IR=K+1
          P=0.0
          DO 10 I=K,N
            Z=A(I,K)
            IF(Z .LT. 0)THEN
              Z=-Z
              ENDIF
            IF(Z .GT. P)THEN
              P=Z
              IP=I
              ENDIF
10          CONTINUE
          IF(P .EQ. 0)THEN
            R=0.0
            RETURN
            ENDIF
          IF(IP .NE. K)THEN
            DO 20 J=K,N
              Z=A(IP,J)
              A(IP,J)=A(K,J)
20            A(K,J)=Z
            Z=X(IP)
            X(IP)=X(K)
            X(K)=Z
            R=-R
            ENDIF
          R=R*A(K,K)
          P=1.0/A(K,K)
          DO 30 J=IR,N
            A(K,J)=A(K,J)*P
            DO 30 I=IR,N
30            A(I,J)=A(I,J)-A(I,K)*A(K,J)
          IF(X(K) .NE. 0)THEN
            X(K)=X(K)*P
            DO 40 I=IR,N
40          X(I)=X(I)-A(I,K)*X(K)
            ENDIF
50        CONTINUE
        IF(A(N,N) .EQ. 0)THEN
          R=0.0
          RETURN
          ENDIF
        R=R*A(N,N)
        X(N)=X(N)/A(N,N)
        DO 70 K=1,M
          I=N-K
          IL=I+1
          S=0.0
          DO 60 L=IL,N
60          S=S+A(I,L)*X(L)
70        X(I)=X(I)-S
        RETURN
        END
```

```
      SUBROUTINE CRVPT(X,Y,L,NO,KO,SH,SWX,JT,MT,JX,MX,JY,MY,IN)
C     X- ARRAY OF X VALUES
C     Y- ARRAY OF Y VALUES
C     L- ARRAY OF POINT CHARACTERS
C           IF MINUS  THEN ONLY A SINGLE CHARACTER
C     N- NUMBER OF DATA POINTS
C     K- REPEAT FACTOR FOR ARRAYS
C     J- BCD ARRAY FOR T- TITLE
C           X- X-AXIS
C           Y- Y-AXIS
C     M- NUMBER OF CHARACTERS IN BCD ARRAYS
C     I- ARRAY OF PLOT TYPES
C           IF  MINUS  THEN A SINGLE INTEGER CONTAINING PLOT TYPES
C
      DIMENSION X(*),Y(*),L(*),IN(*),A(10)
      CHARACTER*80 JT,JX,JY
      CHARACTER*11 IRR
      CHARACTER*3 IZX
      CHARACTER*2 IYEQ
      CHARACTER*1 IZZ,TEKEN
      IRR ='ERROR NO!  '
      IZX ='Z=X'
      IYEQ='Y='
      IZZ ='Z'
      OPEN(UNIT=98,FILE='CRVPLT.DAT')
C...SAME IS USED TO CONCATENATE CALLS TO SYMBOL AND NUMBER
      SAME=999.0
      IM=4
      K=KO
      N=NO
      IN1=IN(1)
      IF(K)1,92,3
1     K=-K
3     NK=N*K
      IF(N)2,92,4
2     N=-N
      NK=N*K
      TH=SH-.5
      GOTO 10
4     TH=SH-1.
C...INITIALIZE, PLOT AXES AND TITLE
      SW=SWX
      IX=SW*4.+1.
      CALL SCALE2(X,SW,N,K)
      XOF=X(NK+1)
      NKK=NK+K+1
      X(NKK)=X(NKK)*2
      XDF=X(NKK)
      CALL SCALE2(Y,SH-3.0,N,K)
      YOF=Y(NK+1)
      YDF=Y(NKK)
      CALL AXIS2(0.0,0.0,JX,-MX,SW,0.0,XOF,XDF)
      CALL AXIS2(0.0,0.0,JY,MY,SH-3.0,90.0,YOF,YDF)
```

```
        CALL SYMBOL(1.27,SH-0.508,.3556,JT,0.0,MT)
C...PLOT DATA POINTS
10      H=0.0
        DO 11 I=1,NK,K
11        H=H+X(I)
        B=N
        H=H/B
        CALL SYMBOL(2.5,TH,0.25,IZX,0.,3)
        IF(H .GT. 0.0)THEN
          TEKEN='-'
         ELSE
          TEKEN='+'
          ENDIF
        WRITE(98,12)TEKEN,H
12      FORMAT(1X,'Z = X ',A1,E20.6)
        CALL FNUM(SAME,TH,.25,-H,IM)
        CALL PLOT(0.0,0.0,-3)
14      TH=TH-.5
        IF(IN1 .LT. 0)THEN
          J=(IN1/10)*10-IN1
          IN1=IN1/10
         ELSE
          J=IN1
          II=2
          ENDIF
        IF(J .EQ. 0)THEN
          MA=0
         ELSE
          MA=2
          ENDIF
        IF(L(1) .LT. 0)THEN
          MA=MA+1
          JA=-L(1)
          ENDIF
        MA=MA+1
        JB=-1
        DO 50 I=1,NK,K
        GOTO(42,44,42,44),MA
42      JA=L(I)
44      XC=(X(I)-XOF)/XDF
        YC=(Y(I)-YOF)/YDF
        IF(XC)78,45,45
45      IF(XC-SWX)47,47,78
47      IF(YC)78,48,48
48      IF(YC-SH)49,49,78
49      CALL SYMBOL(XC,YC,0.2,CHAR(JA),0.0,JB)
        GOTO(46,46,50,50),MA
46      JB=-2
50      CONTINUE
        IF(J)100,60,100
C...TEST FOR POLYNOMIAL FITS
60      IF(IN(1))62,64,64
62      J=(IN1/10)*10-IN1
```

```
        IN1=IN1/10
        GOTO 70
64      J=IN(II)
        II=II+1
70      IF(J)90,90,100
C...ERROR EXIT
78      EN=100+I
80      CALL SYMBOL(SW+3.75,.0,0.5,IRR,0.0,11)
        CALL NUMBER(SAME,0.0,0.5,EN,0.0,-1)
        CALL WHERE(XZ,YZ,ZZ)
        SW=XZ+2.5
C...EXIT
90      EX=0.0
        IF(KO)94,92,92
92      EX=SW+10.
94      CALL PLOT(EX,0.0,-3)
        CLOSE(UNIT=98)
        RETURN
C...FIND AND PLOT A CURVE
100     IF(N-J)102,102,110
102     EN=10+J
        GOTO 80
C...FIND THE COEFFICIENTS
110     CALL CRVFT(X,Y,N,K,J,A,H)
        WRITE(98,113)
113     FORMAT(1X,'Y =')
        DO 111,I=J+1,1,-1
111       WRITE(98,112)A(I),J+1-I
112     FORMAT(1X,E20.6,' * Z^',I1)
        CALL PLOT(0.0,0.0,-3)
        CALL SYMBOL(2.3,TH+0.127,.18,CHAR(J),0.,-1)
        CALL SYMBOL(3.3,TH,0.30,IYEQ,0.0,2)
C...PRINT EQUATION OF CURVE ON GRAPH
114     DO 140 JJ=1,J
        IF(A(JJ))116,140,116
116     CALL FNUM(SAME,TH,.25,A(JJ),IM)
        CALL SYMBOL(SAME,TH,0.30,IZZ,0.0,1)
124     BZ=J-JJ+1
        IF(BZ-1.0)140,140,130
130     CALL NUMBER(SAME,TH+.127,.25,BZ,0.0,-1)
140     CONTINUE
        IF(A(J+1))150,200,150
150     CALL FNUM(SAME,TH,.25,A(J+1),IM)
200     TH=TH-.5
        CALL WHERE(XQ,YQ,ZQ)
        IF(XQ-SW)202,202,201
201     SW=XQ
202     XN=J
C...PLOT ACTUAL CURVE WITH .25 CM  ACCURACY
        XN=XN*.5
        XC=0.0
        IP=3
        DO 250 IT=1,IX
```

```
        XT=XOF+XC*XDF
        YT=A(1)
        DO 220 JJ=1,J
220     YT=YT*(XT-H)+A(JJ+1)
        YC=(YT-YOF)/YDF
        IF(YC)224,230,230
224     YC=0.0
        IP=3
        GOTO 244
230     IF(YC-SH)240,240,234
234     YC=SH
        IP=3
        GOTO 244
240     IF(XC-XN)244,242,242
242     XN=XN+5.0
        CALL SYMBOL(XC,YC,0.15,CHAR(J),0.0,-2)
        GOTO  250
244     CALL PLOT(XC,YC,IP)
        IP=2
250     XC=XC+0.25
        GOTO 60
        END
C
        SUBROUTIE FNUM(X,Y,H,FL,N)
        CHARACTER*3 ITEN
        CHARACTER*4 IZERO
        CHARACTER*1 IPLS,IMIN,IPER
        ITEN ='(10'
        IZERO='+0.0'
        IPLS ='+'
        IMIN ='-'
        IPER ='.'
C
C...SAME IS USED TO CONCATENATE CALLS TO SYMBOL AND NUMBER
        SAME=999.0
        IF(FL)10,20,30
10      F=-FL
        CALL SYMBOL(X,Y,H,IMIN,0.0,1)
        GOTO 40
30      F=FL
        CALL SYMBOL(X,Y,H,IPLS,0.0,1)
40      EF=0.0
        NK=N
        K=N+1
        IF(F-10.0**K)104,102,102
102     EF=K
        F=F/10.0**K
108     IF(F-10.0)200,106,106
106     EF=EF+1.0
        F=F/10.0
        GOTO 108
104     IF(F-10.0)110,112,112
112     K=K-1
```

```
        IF(F-10.0**K)112,114,114
114     NK=NK-K
        GOTO 200
110     IF(F-0.1)116,200,200
116     IF(F-0.01)120,120,118
120     EF=EF-1.0
        F=F*10.0
        IF(F-1.0)120,200,200
118     NK=NK+1
200     CALL NUMBER(SAME,Y,H,F,0.0,NK)
        IF(EF)12,42,12
12      CALL SYMBOL(SAME,Y,H,ITEN,0.0,3)
        IF(EF-1.0)32,44,32
32      CALL NUMBER(SAME,Y+H/2.0,3.0*H/4.0,EF,0.0,-1)
44      CALL SYMBOL(SAME,Y,H,IPER,0.0,1)
42      RETURN
20      CALL SYMBOL(X,Y,H,IZERO,0.0,4)
        RETURN
        END
C
        SUBROUTINE CRVFT(X,Y,NP,K,ND,A,H)
C...CURVE FITTING ROUTINE
C       INPUT-  X,Y  ARRAYS OF POINTS
C               NP   NUMBER OF POINTS
C               K    REPEAT FACTOR
C               ND   DEGREE OF POLYBOMIAL DESIRED
C               H    AVERAGE VALUE OF X,USED FOR POLYNOMIAL
C       OUTPUT- A    ARRAY OF COEFFICIENTS FOR POLYNOMIAL
        DIMENSION X(*),Y(*),A(*),D(10,10)
        NPK=NP*K
        N=ND+1
        DO 40 I=1,N
          A(I)=0.0
          DO 20 J=1,N
20          D(I,J)=0.0
          KP=N-I
          DO 40 L=1,NPK,K
            Z=X(L)-H
            IF(Z .NE. 0)THEN
              A(I)=A(I)+Y(L)*Z**KP
              DO 30 J=1,N
                KR=2*N-I-J
30            D(I,J)=D(I,J)+Z**KR
              ENDIF
40          CONTINUE
C...CALLS SUBROUTINE TO SOLVE MATRIX EQUATION DX=A FOR X AND
C...RETURN SOLUTION IN A WITH DETERMINANT IN R
        CALL MATX(D,A,N,R)
        RETURN
        END
C
        SUBROUTINE MATX(A,X,N,R)
        DIMENSION A(10,10),X(10)
```

```
      R=1.0
      M=N-1
      DO 50 K=1,M
        IR=K+1
        P=0.0
        DO 10 I=K,N
          Z=A(I,K)
          IF(Z .LT. 0)THEN
            Z=-Z
            ENDIF
          IF(Z .GT. P)THEN
            P=Z
            IP=I
            ENDIF
10        CONTINUE
        IF(P .EQ. 0)THEN
          R=0.0
          RETURN
          ENDIF
        IF(IP .NE. K)THEN
          DO 20 J=K,N
            Z=A(IP,J)
            A(IP,J)=A(K,J)
20          A(K,J)=Z
          Z=X(IP)
          X(IP)=X(K)
          X(K)=Z
          R=-R
          ENDIF
        R=R*A(K,K)
        P=1.0/A(K,K)
        DO 30 J=IR,N
          A(K,J)=A(K,J)*P
          DO 30 I=IR,N
30          A(I,J)=A(I,J)-A(I,K)*A(K,J)
        IF(X(K) .NE. 0)THEN
          X(K)=X(K)*P
          DO 40 I=IR,N
40        X(I)=X(I)-A(I,K)*X(K)
          ENDIF
50      CONTINUE
      IF(A(N,N) .EQ. 0)THEN
        R=0.0
        RETURN
        ENDIF
      R=R*A(N,N)
      X(N)=X(N)/A(N,N)
      DO 70 K=1,M
        I=N-K
        IL=I+1
        S=0.0
        DO 60 L=IL,N
60        S=S+A(I,L)*X(L)
```

```
70          X(I)=X(I)-S
          RETURN
          END
```

Graphics and Image Processing
W. Newman* Editor

A Linear Algorithm for Incremental Digital Display of Circular Arcs

Jack Bresenham
IBM System Communications Division

Circular arcs can be drawn on an incremental display device such as a cathode ray tube, digital plotter, or matrix printer using only sign testing and elementary addition and subtraction. This paper describes methodology for producing dot or step patterns closest to the true circle.

Key Words and Phrases: graphics, circle drawing, step generation, dot generation, incremental digital plotting, raster display, integer arithmetic, circle algorithm

CR Categories: 4.41, 8.2

Author's address: IBM System Communications Division, 1000 Westchester Avenue, White Plains, NY 10604.

* Editor when the paper was being processed; J. Foley is now editor of this department.

1. Introduction

This paper describes an algorithm for circular arc mesh point selection using incremental display devices such as a cathode ray tube or digital plotter. Error criteria are explicitly specified and both squared and radial error minimization considered. The repetitive incremental stepping loop for point selection requires only simple addition/subtraction and sign testing; neither quadratic nor trigonometric evaluations are required. When a circle's center point and radius are integers, only integer calculations are required.

The circle algorithm complements an earlier line algorithm described in [1, 2]. The algorithm's minimum error point selection is appropriate for use in numerical control, drafting, or photo mask preparation applications where closeness of fit is a necessity. Its simplicity and use only of elementary addition/subtraction allow its use in small computers, programmable terminals, or direct hardware implementations where compactness and speed are desirable.

The display devices under consideration are capable of executing, in response to an appropriate pulse, any one of the eight linear movements shown in Figure 1. Thus incremental movement is from a point on a mesh to any of its eight adjacent points on the mesh.

All generated data points must lie on mesh points and must consequently have integer display coordinates. It is assumed that by scaling and appropriate translation of axes, the circle is centered at the origin of a rectangular coordinate system whose units are those of the display device.

At each move, the algorithm choses a point so as to minimize the absolute difference between R^2 and the square of the radius to the point. In the Appendix it is shown that this also minimizes the linear difference between R and the radius itself when the circle is centered at a display mesh point and has an integer radius.

In this paper, the algorithm is developed for the case of clockwise movement from $(0, R)$ to $(R, 0)$ through the first quadrant. Requisite modifications for completing the full circular path are then indicated and the basic algorithm stated for tridirectional movement control by quadrant.

For analysis, the first quadrant arc of the circle given by

$$X^2 + Y^2 = R^2, \text{ where } X \triangleq \text{abscissa} \geq 0,\quad Y \triangleq \text{ordinate} \geq 0,\quad R \triangleq \text{an integer} \geq 1$$

will be used. The extension to complete the full circle will then be described as will the modifications required to accomodate an arbitrary arc of the general circle given by

$$(x - a)^2 + (y - b)^2 = r^2$$

with starting point (x_s, y_s) and terminating point (x_t, y_t) specified on circumference.

2. Analysis

In the first quadrant of a circle, y is a monotonically decreasing function of x. Clockwise movement in this quadrant can therefore be accomplished by a sequence of moves involving only m_1, m_2, and m_3.

When the display is at the point P_i, whose coordinates are (X_i, Y_i), the next movement is either m_1 to $(X_i + 1, Y)$ at 0°, m_2 to $(X_i + 1, Y_i - 1)$ at 315°, or m_3 to $(X_i, Y_i - 1)$ at 270°. The absolute difference between R^2 and the squares of the constrained radii of the three alternatives is minimized by determining the minimum of the following quantities:

$$| [(X_i + 1)^2 + Y_i^2] - R^2 |,$$
for m_1 movement to $(X_i + 1, Y_i)$,

$$| [(X_i + 1)^2 + (Y_i - 1)^2] - R^2 |,$$
for m_2 movement to $(X_i + 1, Y_i - 1)$,

$$| [X_i^2 + (Y_i - 1)^2] - R^2 |,$$
for m_3 movement to $(X_i, Y_i - 1)$.

The algorithm simplifies this threefold evaluation to consideration of only two points at each step by first observing the sign of the difference Δ_i:

$$\Delta_i = \{[(X_i + 1)^2 + (Y_i - 1)^2] - R^2\},$$

the difference between R^2 and the square of the radius to the diagonally adjacent point $(X_i + 1, Y_i - 1)$.

Figure 2 illustrates the five possibilities for the circle's intersection with the coordinate lines $X_i + 1$ and $Y_i - 1$ which must be considered when selecting the step for movement from point $P(X_i, Y_i)$. For clarity, subscripts for X_i and Y_i are dropped in developing the alternatives.

(a) If $\Delta_i < 0$, then $(X + 1, Y - 1)$ is in the interior of the true circle, i.e. 1 or 2 in Figure 2. The true circle passes between the points $(X + 1, Y)$ and $(X + 1, Y - 1)$, case 1, or between the points $(X + 1, Y + 1)$ and $(X + 1, Y)$, case 2.

In case 1 the closer of the two points can be found by observing the sign of the difference δ:

$$\delta = | [(X + 1)^2 + Y^2] - R^2 | - | [(X + 1)^2 + (Y - 1)^2] - R^2 |.$$

Since in case 1 the constrained radius to $(X + 1, Y)$ exceeds or equals R while the constrained radius to $(X + 1, Y - 1)$ is less than R,

$$\{[(X + 1)^2 + Y^2] - R^2\} \geq 0 \text{ and}$$
$$\{[(X + 1)^2 + (Y - 1)^2] - R^2\} < 0.$$

Rewriting the definition of δ removing the absolute value expressions thus gives

$$\delta = \{[(X + 1)^2 + Y^2] - R^2\} + \{[(X + 1)^2 + (Y - 1)^2] - R^2\}.$$

Adding and subtracting $2Y - 1$ and collecting terms of $(X + 1)^2$ and $(Y - 1)^2$ gives

$$\delta = 2\{[(X + 1)^2 + (Y - 1)^2] - R^2\} + 2Y - 1.$$

Recalling the definition of Δ_i and substituting it into the previous equation yields

$$\delta = 2\Delta_i + 2Y_i - 1,$$

where $\delta \leq 0$ implies move m_1 and $\delta > 0$ implies move m_2. In case 2 the movement should be m_1. That in case 2, δ is always less than zero and hence forces the required m_1 move can be demonstrated as follows:

$$\delta = 2\Delta_i + 2Y_i - 1,$$
$$\delta = \{[(X + 1)^2 + Y^2] - R^2\} + \{[(X + 1)^2 + (Y - 1)^2] - R^2\}.$$

In case 2 the constrained radii to both $(X + 1, Y)$ and $(X + 1, Y - 1)$ are less than R:

$$\{[(X + 1)^2 + Y^2] - R^2\} < 0 \text{ and}$$
$$\{[(X + 1)^2 + (Y - 1)^2] - R^2\} < 0$$

so that here

$$\delta = -| [(X + 1)^2 + Y^2] - R^2 | - | [(X + 1)^2 + (Y - 1)^2] - R^2 |.$$

Thus $\delta < 0$ in all occurrences of case 2, so m_1 is correctly made.

(b) If $\Delta_i > 0$ then $(X + 1, Y - 1)$ is exterior to the true circle, i.e. cases 3 and 4 in Figure 2. The true circle passes between the points $(X + 1, Y - 1)$ and $(X, Y - 1)$, case 3, or between the points $(X, Y - 1)$ and $(X - 1, Y - 1)$, case 4.

In like manner, an alternate difference δ':

$$\delta' = 2\Delta_i - 2X_i - 1$$

yields a similar selection criterion; where $\delta' \leq 0$ implies move m_2 and $\delta' > 0$ implies move m_3.

(c) If $\Delta_i = 0$ then $(X + 1, Y - 1)$ is on the true circle, i.e. case 5, and the movement should be m_2. In

Fig. 1.

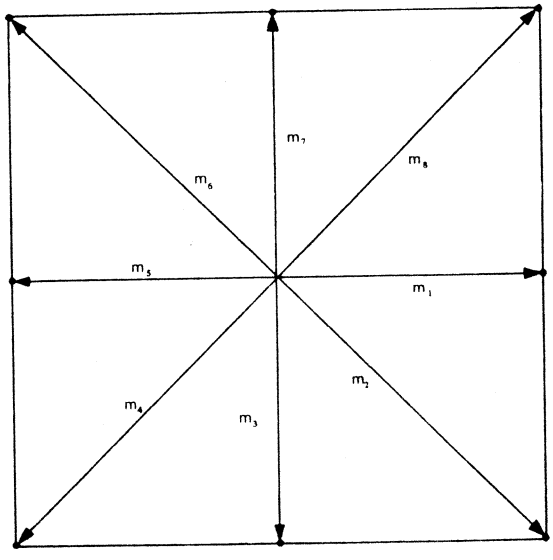
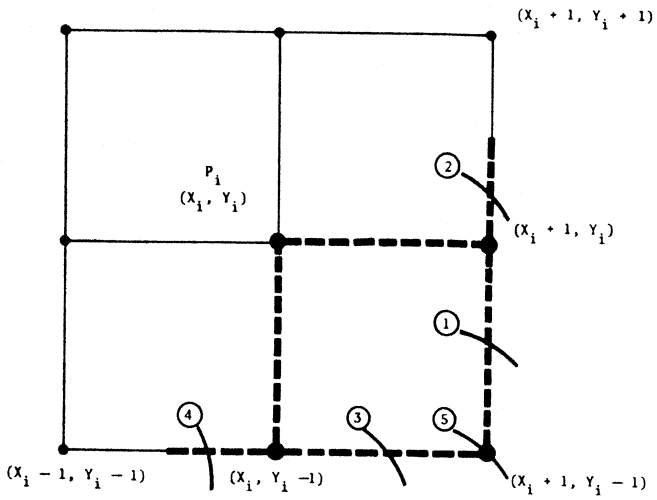


Fig. 2.



this case the above steps yield $\delta > 0$ and $\delta' < 0$ so a proper m_2 move is forced by either calculation.

By noting the expansions

$$(X + 1)^2 = X^2 + 2X + 1, \quad (Y - 1)^2 = Y^2 - 2Y + 1,$$

the difference Δ_i and the coordinates of the display point P_i are observed to have the following recurrence relations:

(i) For m_1 movement (i.e. $\Delta_i < 0$ and $\delta \leq 0$),

$$X_{i+1} = X_i + 1,$$
$$Y_{i+1} = Y_i,$$
$$\Delta_{i+1} = \Delta_i + 2X_{i+1} + 1.$$

(ii) For m_2 movement (i.e. $\Delta_i \leq 0$ and $\delta > 0$, or $\Delta_i \geq 0$ and $\delta' \leq 0$),

$$X_{i+1} = X_i + 1,$$
$$Y_{i+1} = Y_i - 1,$$
$$\Delta_{i+1} = \Delta_i + 2X_{i+1} - 2Y_{i+1} + 2.$$

(ii) For m_3 movement (i.e. $\Delta_i > 0$ and $\delta' > 0$),

$$X_{i+1} = X_i,$$
$$Y_{i+1} = Y_i - 1,$$
$$\Delta_{i+1} = \Delta_i - 2Y_{i+1} + 1.$$

If the circle is started at one of the four intersections of the true circle with the coordinate axes, there is no need to consider any second-order terms in the initial conditions. If the circle starts at the point ($X_0 = 0$, $Y_0 = R$), then

$$\Delta_0 = [(X_0 + 1)^2 + (Y_0 - 1)^2] - R^2 = 2 - 2R$$

and the full first-quadrant clockwise quarter arc will be complete when $Y_i = 0$ or, alternatively, when $Y_i < \frac{1}{2}$.

To complete the remaining three quarter arcs, the movement values m_1, m_2, and m_3 are respecified appropriately; the algorithm is reinitialized and is repeated with the same logic as before until the complete circle is drawn. When crossing a quadrant boundary the algorithm is reinitialized with

$$X_0 \leftarrow 0, \; Y_0 \leftarrow R, \; \Delta_0 \leftarrow 2 - 2R$$

or, alternatively, only current variables can be employed by reinitializing with

$$X_0 \leftarrow -Y_i, \; Y_0 \leftarrow X_i, \; \Delta_0 \leftarrow \Delta_i - 4X_i.$$

As is true for some plotters, let $M1$, $M2$, $M3$ denote, respectively, the appropriate m subscript for actual movement codes within a quadrant corresponding to normalized m_1, m_2, and m_3 first quadrant movement. With initial conditions of $M1_0 = 1$, $M2_0 = 2$, $M3_0 = 3$, one can observe that at each quadrant crossing the clockwise movement code reinitialization is

$$M1_{j+1} \leftarrow M3_j, \; M2_{j+1} \leftarrow M1_{j+1} + 1,$$
$$M3_{j+1} \leftarrow 8 \mid (M1_{j+1} + 2),$$

where $a \mid b$ is b modulo a.

Alternatively, movement can be coded as a 2-tuple of delta abscissa and ordinate (Δx, Δy) increments 1, 0, or -1. With initial conditions of $M1_0 = (1, 0)$, $M2_0 = (1, -1)$, and $M3_0 = (0, -1)$, one can observe that at each quadrant crossing the clockwise movement code reinitialization is

$$M1_{j+1} \leftarrow M3_j, \; M2_{j+1} \leftarrow (M2[2]_j, -M2[1]_j),$$
$$M3_{j+1} \leftarrow (M3[2]_j, -M3[1]_j).$$

To adapt the above basic algorithm[1] for arbitrary circular arcs having noninteger radii and center points, initialization must be considered in more detail. The general circle will be $(x - a)^2 + (y - b)^2 = r^2$ with a starting point (x_s, y_s) and terminating point (x_t, y_t) given on the circumference. Direction of rotation will be clockwise ($D = 1$) or counterclockwise ($D = -1$).

As the path to be calculated must consist of integer mesh points, it is necessary to determine the mesh points closest to the starting and terminating points. Any point (x, y) on the circumference lies within a unit square having mesh point corners of

$$C: \{(\lfloor x \rfloor, \lfloor y \rfloor) (\lfloor x \rfloor, \lceil y \rceil) (\lceil x \rceil, \lfloor y \rfloor) (\lceil x \rceil, \lceil y \rceil)\}$$

where $\lfloor x \rfloor$ is the greatest integer equal to or less than x and $\lceil x \rceil$ is the least integer equal to or greater than x. If x and/or y are integers, the unit square degenerates to a single point or a unit length line and the four member set C will contain only one or two unique elements. The closest mesh point (X', Y') is that point from C which minimizes the difference:

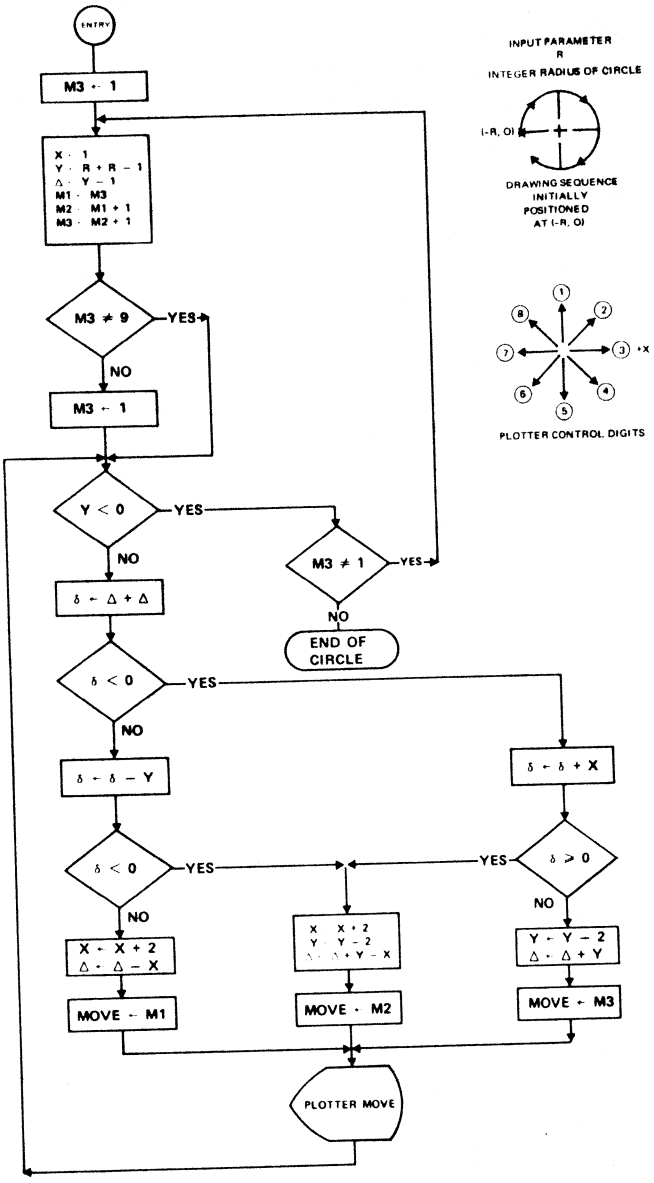
$$|(x' - a)^2 + (y' - b)^2 - r^2| \quad (x', y') \ni C.$$

Translating the closest mesh points to an origin coincident with the center of the circle then gives closest starting and terminating points of

$$\hat{X}_s = X_s' - a, \; \hat{Y}_s = Y_s' - b,$$
$$\hat{X}_t = X_t' - a, \; \hat{Y}_t = Y_t' - b.$$

[1] A flow chart for the basic integer, full circle case [3] which takes advantage of the control format codes for incremental digital plotting with the IBM 1627 is shown in Figure 3.

Fig. 3.



Since all calculations are based upon a standard first quadrant clockwise case, it is necessary to transform any other situation to the normalized case. Table I gives the transformation to obtain (X_s, Y_s) and (X_t, Y_t) from $(\hat{X}_s, \hat{Y}_s)$ and $(\hat{X}_t, \hat{Y}_t)$. Table I also gives a quadrant indicator, q, from which the number of quadrant crossings between the two points can be determined together with the initial movement codes associated with an arbitrary point.

The number of quadrant crossings is first calculated as follows:

$$Q^* = 4 \mid (q_t - q_s),$$

then, to consider $Q^* = 0$, the possible ambiguity is resolved by:

(a) $Q = Q^* - 1$ if $Q^* \neq 0$, or if $Q^* = 0$ and either $X_t \geq X_s$ and $Y_t < Y_s$ or $X_t > X_s$ and $Y_t \leq Y_s$;

(b) $Q = 3$ otherwise (i.e. $Q^* = 0$ and $X_t \leq X_s$ and $Y_t \geq Y_s$)

For simplicity, the above defaults to a full circle the case when the starting and terminating points differ yet have the same closest mesh point.

The final point has been selected when no further quadrant crossings remain (i.e. $Q_j < 0$) and $X_t \leq X_i$ and $Y_t \geq Y_i$.

3. Algorithm

Given the arc's starting and terminating points, (x_s, y_s) and (x_t, y_t), on the circle circumference together with the circle's center point (a, b) and direction of rotation (D), the display mesh point selection algorithm can be summarized as follows for the general case:

Notation

X_i is the "constrained circle's" translated abscissa value at the ith step of normalized clockwise movement in the first quadrant.

Y_i is the "constrained circle's" translated ordinate value at the ith step of normalized clockwise movement in the first quadrant.

Q_j is the number of quadrants remaining to be traversed.

$M1$ represents relative 0° movement for normalized clockwise, first-quadrant incrementation.

$M2$ represents relative 315° movement for normalized clockwise, first-quadrant incrementation.

$M3$ represents relative 270° movement for normalized clockwise, first-quadrant incrementation.

Δ_i is the signed difference $\{[(X_i+1)^2 + (Y_i-1)^2] - R^2]\}$. The sign of Δ_i indicates whether the point (X_i+1, Y_i-1) is inside or outside of the true circle.

δ is the signed difference $\{[(X_i+1)^2 + Y_i^2] - R^2\} + \Delta_i$. The sign of δ indicates which of the two points (X_i+1, Y) or (X_i+1, Y_i-1) is closest to the true circle.

δ' is the signed difference $\{[X_i+(Y_i-1)^2] - R^2\} + \Delta_i$. The sign of δ' indicates which of the two points (X_i, Y_i-1) or (X_i+1, Y_i-1) is closest to the true circle.

Initialization

1. Determine closest mesh points (X_s', Y_s') and (X_t', Y_t') from (x_s, y_s) and (x_t, y_t) by finding their respective unit-square corner mesh points from C_s and C_t which minimize

   $$\mid [(x'-a)^2 + (y'-b)^2] - [(x-a)^2 + (y-b)^2] \mid$$

   from $C(x', y')$: $\{([x], [y])\ ([x], [y])\ ([x], [y])\ ([x], [y])\}$.

2. Translate to zero centered circle coordinates:

   $$\hat{X}_s = X_s' - a,\ \hat{Y}_s = Y_s' - b,\ \hat{X}_t = X_t' - a,\ \hat{Y}_t = Y_t' - b.$$

3. Transform $(\hat{X}_s, \hat{Y}_s)$ and $(\hat{X}_t, \hat{Y}_t)$ to normalized, first quadrant, clockwise coordinates per Table I to determine

   $$X_s = X_0,\ Y_s = Y_0,\ q_s,\ M1_s = M1_0,\ M2_s = M2_0,\ M3_s = M3_0, \text{ and } X_t,\ Y_t,\ q_t.$$

4. Calculate the number of quadrant crossings from $Q^* = 4 \mid (q_t - q_s)$ as

   $$Q_0 = 3, \text{ if } Q^* = 0 \text{ and } X_t \leq X_s \text{ and } Y_t \geq Y_s,$$
   $$Q_0 = Q^* - 1, \text{ otherwise.}$$

5. Calculate the initial decision difference Δ_0 as

$$\Delta_0 = [(X_s+1)^2 + (Y_s-1)^2] - [(x_s-a)^2 + (y_s-b)^2]$$
$$= \{[(X_s'-a)^2 + (Y_s'-b)^2] - [(x_s-a)^2 + (y_s-b)^2]\} + 2(X_s-Y_s+1).$$

6. Set the direction of rotation indicator as

$D = 1$, for clockwise rotation,
$D = -1$, for counterclockwise rotation.

Incremental Stepping Loop

1. If $Q_j \geq 0$, i.e. not in final quadrant, go to step 2. Otherwise:
   a. If $X_t > X_i$ or $Y_t < Y_i$, go to step 2.
   b. Otherwise, terminate.

2. If $Y_i \geq \frac{1}{2}$, i.e. quadrant not complete, go to step 3. Otherwise, reinitialize to

$$X_0 \leftarrow -Y_i, \quad Y_0 \leftarrow X_i, \quad \Delta_0 \leftarrow \Delta_i - 4X_i, \quad Q_{j+1} \leftarrow Q_j - 1$$

$$M1_{j+1} \leftarrow M3_j, \quad M2_{j+1} \leftarrow D\cdot(M2[2]_j, -M2[1]_j),$$
$$M3_{j+1} \leftarrow D\cdot(M3[2]_j, -M3[1]_j)$$

and return to step 1.

3. a. If $\Delta_i \leq 0$, calculate

$$\delta \leftarrow 2\Delta_i + 2Y_i - 1,$$

and if $\delta \leq 0$ move $M1$, if $\delta > 0$ move $M2$.
   b. If $\Delta_i > 0$ calculate

$$\delta' \leftarrow 2\Delta_i - 2X_i - 1;$$

and if $\delta' \leq 0$ move $M2$, if $\delta' > 0$ move $M3$.

4. a. If movement was $M1$, then

$$X_{i+1} \leftarrow X_i + 1, \quad Y_{i+1} \leftarrow Y_i, \quad \Delta_{i+1} \leftarrow \Delta_i + 2X_{i+1} + 1.$$

   b. If movement was $M2$, then

$$X_{i+1} \leftarrow X_i + 1, \quad Y_{i+1} \leftarrow Y - 1,$$
$$\Delta_{i+1} \leftarrow \Delta_i + 2X_{i+1} - 2Y_{i+1} + 2.$$

   c. If movement was $M3$, then

$$X_{i+1} \leftarrow X_i, \quad Y_{i+1} \leftarrow Y_i - 1, \quad \Delta_{i+1} \leftarrow \Delta_i - 2Y_{i+1} + 1.$$

5. Return to step 1.

4. Remarks

Other incremental algorithms for displaying figures have been described elsewhere [1–12]. Pitteway [10] first published a general solution for simply displaying conic sections incrementally by differencing the general polynominal and using a bi-directional movement control by octant. In return for only a very slight overhead when crossing the additional four 45° octant boundaries, his bi-directional control methodology offers the most efficient inner stepping loop of any algorithm of which the author is aware. A variant of his bi-directional method for vertical error minimization can be used with the clockwise, first quadrant normalization technique described here for squared error minimization to also achieve a three addition inner loop for both square and diagonal moves. From 90° to 45° one uses the two variables $2(X - Y) + 1$ and $2X + 1$, then from 45° to 0° one tracks the variables $2(Y - X) - 1$ and $2Y - 1$. The square movement code is changed at 45° while the diagonal move is changed at 0° (i.e. at $2(X - Y) + 1 > 0$ and at $2Y - 1 < 0$). For implementation symmetry, the decision difference, d, is $d = \frac{1}{2}\delta$ between 90° and 45° and is $d = -\frac{1}{2}\delta'$ between 45° and 0°.

Addressing each conic section separately, Metzger [9] provided a set of compact algorithms for incremental display, though, in all but the straight line case, quadratic or square root calculation is required.

Jordan, Lennon, and Holm [8] have unified with very good clarity a generalized but efficient solution for incrementally displaying, with arbitrary step sizes (Δx, Δy) explicitly covered, any curve possessing continuous derivatives. As special cases, they describe a tri-directional movement control polynominal display algorithm functionally comparable to that of Pitteway and a circle display algorithm functionally comparable to the one presented here. Though comparable, the Jordan and Pitteway algorithms do differ in error criteria in that Jordan minimizes function residue or, for circles, squared error while Pitteway minimizes vertical or horizontal error.

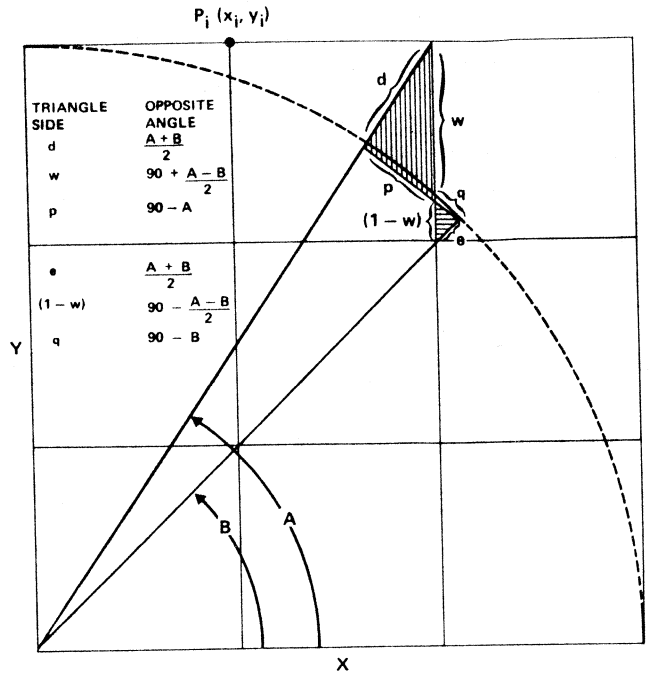
A linear algorithm using only integer calculation has been described by Denert [7] for polygon approximation of circles. In two short notes [11, 12] Pitteway has briefly discussed variants and an alternative to the Denert algorithm using [1, 2, 10].

Cohen [4, 5] has presented a method of generating a sequence of regularly spaced points on a circle using the iteration $P_{(i+1)} = TP_i$ where T is a 2×2 matrix. Successive points then are connected by straight line segments to approximate the curve. For other conic sections, the method generates variable density points

Table I. Transformation Table: Normalize to Standard Clockwise First Quadrant Case.

Quadrant	Rotation	$\hat{X}$	$\hat{Y}$	Index	X	Y	q	$M1$	$M2$	$M3$
III	CCW	<0	<0	0	$\lvert\hat{Y}\rvert$	$\lvert\hat{X}\rvert$	3	0, −1	1, −1	1, 0
II	CCW	<0	≥0	1	$\lvert\hat{X}\rvert$	$\lvert\hat{Y}\rvert$	2	−1, 0	−1, −1	0, −1
IV	CCW	≥0	<0	2	$\lvert\hat{X}\rvert$	$\lvert\hat{Y}\rvert$	0	1, 0	1, 1	0, 1
I	CCW	≥0	≥0	3	$\lvert\hat{Y}\rvert$	$\lvert\hat{X}\rvert$	1	0, 1	−1, 1	−1, 0
III	CW	<0	<0	4	$\lvert\hat{X}\rvert$	$\lvert\hat{Y}\rvert$	2	−1, 0	−1, 1	0, 1
II	CW	<0	≥0	5	$\lvert\hat{Y}\rvert$	$\lvert\hat{X}\rvert$	3	0, 1	1, 1	1, 0
IV	CW	≥0	<0	6	$\lvert\hat{Y}\rvert$	$\lvert\hat{X}\rvert$	1	0, −1	−1, −1	−1, 0
I	CW	≥0	≥0	7	$\lvert\hat{X}\rvert$	$\lvert\hat{Y}\rvert$	0	1, 0	1, −1	0, −1

Fig. 4.



with points most closely spaced in those portions of the curve having the greatest curvature.

In personal communications [March and April, 1975], Pitteway has shown the author an alternative interpretation which provides additional insight into the difference between squared and vertical error minimization. The expressions for δ and δ' can be refactored as follows:

$$\delta = 2\{(X_i + 1)^2 + (Y_i - \tfrac{1}{2})^2 - (r^2 - \tfrac{1}{4})\},$$
$$\delta' = 2\{(X_i + \tfrac{1}{2})^2 + (Y_i - 1)^2 - (r^2 - \tfrac{1}{4})\},$$

which is twice the residue of a circle of radius $(r^2 - \frac{1}{4})^{\frac{1}{2}}$ evaluated respectively at the points $(X + 1, Y - \frac{1}{2})$ and $(X + \frac{1}{2}, Y - 1)$. The algorithm essentially is using the decision rule for squared error minimization:

If $\delta \leq 0$, move m_1
  If $\delta' > 0$, move m_3
    Otherwise, move m_2

The Pitteway algorithm, applied to circles, effectively evaluates these "step and a half" residues using a radius of r rather than $(r^2 - \frac{1}{4})^{\frac{1}{2}}$ and applies the same decision rule. In implementation, the difference between vertical and squared criteria amounts only to a bias of $\frac{1}{4}$ in the initial value of δ.

When a circle's center point and radius are limited to only integer values, identical display points will be selected regardless of whether one minimizes squared, vertical, or radial error. Depending upon which error criterion is used, different display points occasionally will be selected in the general noninteger case. The two circles $(x - 4.53)^2 + (y + 3.6)^2 = (10)^2$ and $x^2 + y^2 = (4.925)^2$ illustrate the differences. In the first example, clockwise movement from (14, −2) will be to (15, −3) by either vertical or radial criteria but will be to (14, −3) by the squared criteria of the Jordan or Bresenham algorithms. In the second example, clockwise movement from (1, 5) will be to (2, 4) by either squared or radial criteria but will be to (2, 5) by the vertical criteria of the Pitteway or Metzger algorithms.

Appendix

It remains to be shown that minimizing the difference between the squares of the true and constrained radii also minimizes the linear difference between the two radii. Figure 4 shows the case in which the circle passes between the points $(X + 1, Y)$ and $(X + 1, Y - 1)$.

Using the notation in Figure 4, application of the trigonometric law of sines for p and q and the law of cosines for d^2 and e^2 gives

$$d^2 = w^2 + p^2 - 2pw \cos \tfrac{1}{2}(A + B),$$

$$d^2 = w^2 + p\left[w\frac{\sin (90 - A)}{\sin [90 + \frac{1}{2}(A - B)]} - 2w \cos \tfrac{1}{2}(A + B)\right]$$

$$d^2 = w^2 + pw\left[\frac{\cos A - 2 \cos \frac{1}{2}(A + B) \cos \frac{1}{2}(A - B)}{\cos \frac{1}{2}(A - B)}\right],$$

$$d^2 = w^2 - pw \frac{\cos B}{\cos \frac{1}{2}(A - B)},$$

$$d^2 = w^2\left[1 - \frac{\cos A \cos B}{\cos^2 \frac{1}{2}(A - B)}\right]. \tag{1}$$

$$e^2 = (1 - w)^2 + q^2 - 2q(1 - w) \cos \tfrac{1}{2}(A + B),$$

$$e^2 = (1 - w)^2 + q\left\{(1 - w)\frac{\sin (90 - B)}{\sin [90 - \frac{1}{2}(A - B)]} - 2(1 - w) \cos \tfrac{1}{2}(A + B)\right\},$$

$$e^2 = (1 - w)^2 + q(1 - w) \cdot \left[\frac{\cos B - 2 \cos \frac{1}{2}(A + B) \cos \frac{1}{2}(A - B)}{\cos \frac{1}{2}(A - B)}\right],$$

$$e^2 = (1 - w)^2 - q(1 - w)\left[\frac{\cos A}{\cos \frac{1}{2}(A - B)}\right],$$

$$e^2 = (1 - w)^2\left[1 - \frac{\cos A \cos B}{\cos^2 \frac{1}{2}(A - B)}\right]. \tag{2}$$

From (1) and (2)

$$d^2 + e^2 = [w^2 + (1 - w)^2]\left[1 - \frac{\cos A \cos B}{\cos^2 \frac{1}{2}(A - B)}\right] \tag{3}$$

Now $90° > A > B \geq 0°$ and $1 \geq w \geq 0$. Hence

$$1 \geq [w^2 + (1 - w)^2] \geq \tfrac{1}{2}$$

and

$$1 > \left[1 - \frac{\cos A \cos B}{\cos^2 \frac{1}{2}(A - B)}\right] > 0$$

so that

$$1 > d^2 + e^2 > 0. \tag{4}$$

In a like manner, (4) can be derived for the δ' situation in which the point $(X + 1, Y - 1)$ and the line segment d' are exterior to the circle and the point $(X, Y - 1)$ and the line segment e' are interior to the circle.

Now

$$\delta = \{[(X+1)^2 + Y^2] - R^2\} + \{[(X+1)^2 + (Y-1)^2] - R^2\}.$$

Hence from Figure 4

$$\delta = [(R+d)^2 - R^2] + [(R-e)^2 - R^2],$$
$$\delta = 2R(d-e) + (e^2 + d^2),$$

so that

$$d^2 + e^2 = \delta - 2R(d-e). \tag{5}$$

From (4) and (5) then

$$1 > \delta - 2R(d-e) > 0$$

and

$$\delta/2R > (d-e) > (\delta - 1)/2R. \tag{6}$$

Therefore

$$\text{if } \delta \leq 0 \text{ then } (d-e) < 0 \text{ hence } d < e, \tag{7}$$
$$\text{if } \delta \geq 1 \text{ then } (d-e) > 0 \text{ hence } d > e. \tag{8}$$

When δ is integer valued only, one has the simple decision rule

$$\text{if } \delta \leq 0 \text{ then } d < e, \tag{9}$$
$$\text{if } \delta > 0 \text{ then } d > e. \tag{10}$$

In a like manner, (9) and (10) can be shown to hold for the δ' situation in which the point $(X + 1, Y - 1)$ and line segment d' are exterior to the true circle.

The circle algorithm thus minimizes both the linear difference between the true and constrained radii and the difference between the squares of the true and constrained radii when the circle has an integer radius and integer center point. From eqs. (4) and (5) and the sum of the square roots of eqs. (1) and (2) (i.e. $0 < d + e < 1$), it can be shown that if $\delta \leq 0$ then $d < \frac{1}{2}$, and if $\delta \geq 1$ then $e < \frac{1}{2}$, which agrees with the maximum radial error found experimentally in [8] and [9].

When δ can assume noninteger values in the range $0 < \delta < 1$, radial and squared criteria need not coincide as one quickly can verify when stepping clockwise from (1, 5) on the circle $x^2 + y^2 = (4.93)^2$. Should decision rule (10) be used when $0 < \delta < 1$, e will be selected when in fact d possibly could be the lesser of the two radial error measures. The difference should be negligible for larger radii since, for $0 < \delta < 1$, one can observe from equations (1), (2), (4), and (5) that $e < 1/2 + 1/4r$ and $|d - e| < 1/2r$.

When incrementally displaying the circle $(x - 4.53)^2 + (y + 3.6)^2 = (10)^2$, movement clockwise from (14, −2) or counterclockwise from (15, −4) to the point (14, −3) provides an example of minimum squared error coinciding with a nonminimum radial error, or normal distance to the curve, in excess of $\frac{1}{2}$ unit. At (14, −3) the squared error is $\simeq$ 9.96 while radial error is $\simeq$0.511. At (15, −3) the squared error is $\simeq$9.98 while radial error is $\simeq$ 0.487. One also can observe in this case that the selected points for the full circle do not exhibit the quadrant to quadrant (or octant to octant) symmetry found when a circle's center point and radius are integers.

Acknowledgments. The suggestions, comments and insights of M.L.V. Pitteway, W.M. Newman, and the ACM referees were of considerable assistance. Helpful also were various discussions with D. van Alderwerelt, F.K. Allen, H.P. Barnett, B.M. Burke, D. Clark, and C.L. Fisher of IBM. R.L. Pratt's June 1965 communication concerning the comparable forcing situation overlooked in [2] led to recognition of cases 2 and 4 shown in Figure 2. W.M. Newman suggested Figure 2 during the review process.

Received June 1974; revised September 1975

References

1. Bresenham, J.E. An incremental algorithm for digital plotting. Presented at ACM Nat. Conf. (Aug. 1963).
2. Bresenham, J.E. Algorithm for computer control of a digital plotter. *IBM Systems J. 4*, 1 (1965), 25–30.
3. Bresenham, J.E. A linear, incremental algorithm for digitally plotting circles. Tech. Rep. No. TR02.286, IBM General Products Div., San Jose, Calif. Jan 27, 1964.
4. Cohen, D. On linear difference curves. Proc. Int. Symp. CG-70, Vol. I, Brunel U. Uxbridge, England, April 1970.
5. Cohen, D. Incremental methods for computer graphics. Tech. Rep. ESD-TR69-193 Harvard U., Cambridge, Mass., April 1969.
6. Danielsen, P.E. Incremental curve generation. *IEEE Trans. Computers C-19*, 9 (Sept. 1970), 783–793.
7. Denert, E. A method for computing points on a circle using only integers. *Comptr. Graphics and Image Processing 2*, 1 (Aug. 1973), 83–91.
8. Jordan, B.W., Lennon, W.J., and Holm, B.C. An improved algorithm for the generation of nonparametric curves. *IEEE Trans. Computers C-22*, 12 (Dec. 1973), 1052–1060.
9. Metzger, R.A. Computer generated graphic segments in a raster display. Proc. AFIPS 1969 SJCC, AFIPS Press, Montvale, N.J., pp. 161–172.
10. Pitteway, M.L.V. Algorithm for drawing ellipses or hyperbolae with a digital plotter. *Comptr. J. 10*, 3 (Nov. 1967), 282–289.
11. Pitteway, M.L.V. Integer circles, etc.—three move extension of Bresenham's algorithm. *Comptr. Graphics and Image Processing 3*, 3 (Sept. 1974), 260–261.
12. Pitteway, M.L.V. Integer circles—some further thoughts. *Compt. Graphics and Image Processing 3*, 3 (Sept. 1974), 262–265.

Graphics and Image Processing
W. Newman* Editor

A Linear Algorithm for Incremental Digital Display of Circular Arcs

Jack Bresenham
IBM System Communications Division

Circular arcs can be drawn on an incremental display device such as a cathode ray tube, digital plotter, or matrix printer using only sign testing and elementary addition and subtraction. This paper describes methodology for producing dot or step patterns closest to the true circle.

Key Words and Phrases: graphics, circle drawing, step generation, dot generation, incremental digital plotting, raster display, integer arithmetic, circle algorithm

CR Categories: 4.41, 8.2

Author's address: IBM System Communications Division, 1000 Westchester Avenue, White Plains, NY 10604.

* Editor when the paper was being processed; J. Foley is now editor of this department.

1. Introduction

This paper describes an algorithm for circular arc mesh point selection using incremental display devices such as a cathode ray tube or digital plotter. Error criteria are explicitly specified and both squared and radial error minimization considered. The repetitive incremental stepping loop for point selection requires only simple addition/subtraction and sign testing; neither quadratic nor trigonometric evaluations are required. When a circle's center point and radius are integers, only integer calculations are required.

The circle algorithm complements an earlier line algorithm described in [1, 2]. The algorithm's minimum error point selection is appropriate for use in numerical control, drafting, or photo mask preparation applications where closeness of fit is a necessity. Its simplicity and use only of elementary addition/subtraction allow its use in small computers, programmable terminals, or direct hardware implementations where compactness and speed are desirable.

The display devices under consideration are capable of executing, in response to an appropriate pulse, any one of the eight linear movements shown in Figure 1. Thus incremental movement is from a point on a mesh to any of its eight adjacent points on the mesh.

All generated data points must lie on mesh points and must consequently have integer display coordinates. It is assumed that by scaling and appropriate translation of axes, the circle is centered at the origin of a rectangular coordinate system whose units are those of the display device.

At each move, the algorithm choses a point so as to minimize the absolute difference between R^2 and the square of the radius to the point. In the Appendix it is shown that this also minimizes the linear difference between R and the radius itself when the circle is centered at a display mesh point and has an integer radius.

In this paper, the algorithm is developed for the case of clockwise movement from (0, R) to (R, 0) through the first quadrant. Requisite modifications for completing the full circular path are then indicated and the basic algorithm stated for tridirectional movement control by quadrant.

For analysis, the first quadrant arc of the circle given by

$$X^2 + Y^2 = R^2, \text{ where } X \triangleq \text{abscissa} \geq 0,\quad Y \triangleq \text{ordinate} \geq 0,\quad R \triangleq \text{an integer} \geq 1$$

will be used. The extension to complete the full circle will then be described as will the modifications required to accomodate an arbitrary arc of the general circle given by

$$(x - a)^2 + (y - b)^2 = r^2$$

with starting point (x_s, y_s) and terminating point (x_t, y_t) specified on circumference.

HIGH-RESOLUTION PRINTER GRAPHICS

BY MARK BRIDGER AND MARK GORESKY

You can address the individual dots used to generate dot-matrix characters

ONE OF THE GREATEST frustrations in doing graphics on a microcomputer is the rather low resolution of the usual microcomputer monitor. The standard IBM Personal Computer color-graphics adapter and monitor display a maximum screen size of 640 by 200 pixels (picture elements); other computers and configurations do somewhat better, perhaps as much as 720 by 350 pixels. It is difficult to draw horizontal lines fast enough to keep the image from flickering. And there are limits to the amount of screen memory available on standard graphics boards.

Many dot-matrix printers are capable of printing individual dots at a much higher resolution than the typical CRT (cathode-ray tube) screen can display them. The Epson FX-80 and the IBM graphics printer are capable, for example, of printing 240 dots per inch horizontally (1920 dots per line) and 216 dots per inch vertically—the latter by printing a line of graphics, advancing the paper one-third of a dot, printing another "interlaced" line of graphics, etc. Other printers can perform similar feats. To use this capability you need to figure out how to "fire the pins," and you need enough extra memory to record where all the dots are to go. This article will show you how to draw some lines and curves on your printer with a resolution of up to 1600 by 640 dots.

SETTING UP THE "PRINTER SCREEN"

The first problem is memory. If you think of a dot as being either on or off, to use an analogy with the screen display, then encoding 1600 by 640 dots, or 1,024,000 points, requires that many *bits* of information. If you divide by 8 to convert bits to bytes, then the process requires 128,000 bytes, or nearly 128K bytes of memory. Somehow, you must set aside that much memory to record this image. Unfortunately, this is not easily done in BASIC, so we must look elsewhere.

The most widely used microcomputer language that allows fencing off this much memory is Pascal, and because Turbo Pascal lets you point to nearly all available memory without having to give explicit addresses, it is the easiest language to use.

Let's set up two 64K-byte memory areas that represent the even lines and the odd lines of a picture. Each of these areas is represented by the following Pascal data type:

```
Type data_type = array[0..1599,
  0..39] of char;
```

This type of variable is a doubly indexed 1600 by 40 array of characters; since one byte represents each character, this multiplies to about 64K bytes.

Now let's declare the variables that are to reserve this space:

```
Var Evenmap, Oddmap:
  ^data_type;
```

(continued)

Mark Bridger and Mark Goresky are associate professors of mathematics at Northeastern University. Mark Bridger has a Ph.D. from Brandeis University; Mark Goresky holds a Ph.D. from Brown University. Mark Bridger can be reached at Bridge Software, 31 Champa St., Newton Upper Falls, MA 02164. Mark Goresky can be reached at the Mathematics Dept., Northeastern University, 360 Huntington Ave., Boston, MA 02115.

The "^" defines a pointer. When you actually create these variables during program execution, using the command New, the computer sets aside two blocks of free memory and automatically reserves them for your use. Each of the variables Evenmap and Oddmap "points" to the beginning of one of these blocks, and you need never concern yourself with exactly where in memory these blocks reside.

How a Dot-Matrix Printer Draws Dots

The print head of a dot-matrix printer normally has seven or more wires, arranged vertically; the most common number is nine. (Eight are used to draw most of the characters, while the ninth is used to draw the bottoms of the g and y characters and to underline.) When typing letters, the printer receives the ASCII code of the character—a number between 0 and 255. As the print head moves across the page, it extends certain wires, depending on the pattern stored in the printer's memory for that character, and the head strikes them against the paper. Usually from 9 to 12 such columns of dots are needed to make a character.

You want to be able to tell the printer directly which wires to fire; in other words, you want to bypass that part of the printer's memory that stores the patterns for the printing of usual characters (letters, numbers, etc.)—you want to do *bit-mapped* graphics. Most printers support this; it is usually called graphics mode. Let's try to address a particular dot on the page.

First, since the wires on the print head are not that close together, you can make use of tiny partial linefeeds to double the number of vertical dots. Table 1 contains a diagram of how it works. The characters represent dot positions on the page; the 1s represent the dots that you actually want to draw and the 0s represent the dot positions you want to skip. To get maximum resolution, you want the dots to be as close to each other as possible, both horizontally and vertically. Getting them close horizontally is accomplished by means of a simple printer command. To get them close vertically, you must divide the picture into the even rows (0, 2, 4, etc.) and the odd rows (1, 3, 5, etc.), as shown in table 2.

When the printer is in graphics mode, the printer prints, for each byte you send it, any pattern of eight vertical dots you specify. The strategy in table 2 is to do the following:

1. Send the printer the 10 bytes that specify the 10 columns represented by the even rows.
2. Instruct the printer to do a carriage return plus a linefeed of one-half a vertical dot.
3. Send the printer the 10 bytes that specify the 10 columns represented by the odd rows.
4. Instruct the printer to do a carriage return plus a linefeed of 7½ vertical dots, preparing it to draw more sets of even and odd rows if there are any.

In more ambitious applications you can have as many as 1600 columns across instead of just these 10. The array pointers Evenmap and Oddmap store this information for the printer. Each represents 1600 columns; each column is 40 bytes or 320 dots high. Looked at another way, there are 320

(continued)

Table 1: *This table shows the dot positions on the page. The 1s represent dots that you actually want to draw; the 0s, dot positions you want to skip over.*

0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	1
11	0	0	0	0	0	0	0	0	1	0
12	0	0	0	0	0	0	0	1	0	0
13	0	0	0	0	0	0	1	0	0	0
14	0	0	0	0	0	1	0	0	0	0
15	0	0	0	0	1	0	0	0	0	0

Table 2: *This table shows the distribution of the various print dot positions between even and odd rows.*

Even:		Odd:
0	1 0 0 0 0 0 0 0 0 0 0	
		1 0 1 0 0 0 0 0 0 0 0
2	0 0 1 0 0 0 0 0 0 0	
		3 0 0 0 1 0 0 0 0 0 0
4	0 0 0 0 1 0 0 0 0 0	
		5 0 0 0 0 0 1 0 0 0 0
6	0 0 0 0 0 0 1 0 0 0	
		7 0 0 0 0 0 0 0 1 0 0
8	0 0 0 0 0 0 0 0 1 0	
		9 0 0 0 0 0 0 0 0 0 1
10	0 0 0 0 0 0 0 0 0 1	
		11 0 0 0 0 0 0 0 0 1 0
12	0 0 0 0 0 0 0 1 0 0	
		13 0 0 0 0 0 0 1 0 0 0
14	0 0 0 0 0 1 0 0 0 0	
		15 0 0 0 0 1 0 0 0 0 0

even rows and 320 odd rows. Each row is 1600 dots wide, and the printer will print eight even or eight odd rows in each pass. Note that these rows form a natural unit totaling 16 rows; let's call such a unit a printer line.

How to Locate a Dot on the Page

Let's write a procedure—Pset(x,y, color)—that draws a point of coordinates x and y in the proper place in one of the two arrays. The coordinates x and y denote the point's column and row (measured from the upper left-hand corner), respectively. The variable color can be equal to either 0 or 1: 0 means erase any point existing at that location; 1 means insert a point there. [*Editor's note: All programs shown here are available for downloading on* BYTE*net Listings. Before November 1 call* (617) 861-9774. *Afterwards, call* (617) 861-9764.]

See listing 1 for the procedure Pset. Start at the line that reads color := color mod 2. First the procedure ensures that color is in the correct range by applying a mod 2 to it. (When K and N are whole numbers, K mod N finds the remainder you get when you divide K by N. When you divide by 2 you can get a remainder of only 0 or 1, depending on whether K is even or odd, respectively.) Next, you determine which printer line you're in by dividing the row number by 16 (y div 16). When you know this line number, you can determine which vertical dot *within* that line you're in; this is *height*. Finally, y mod 2 tells you whether your dot is in an even or an odd row.

For example, suppose you want to print a dot in column 1173, row 554. Then x equals 1173 and y equals 554. 554 div 16 equals 34, so you are in the 34th printer line. 554 mod 16 is 10 and 10 div 2 is 5, so the height of the dot within the printer line is 5; since 554 is even, you are in the array pointed to by Evenmap. The program now calls on the procedure Change to insert this point into the correct position in memory.

The problem now, and the reason Change is so complicated, is that

(*continued*)

Listing 1: *Epson* FX-80 *procedures in disk file* Printpak.pas.

```
const
  across = 1599;     (** replace with 1249 for Prowriter **)
  down   = 39;
type
  data_type = array[0..across, 0..down] of char;
  mask_array = array[0..7] of byte;
var
  Evenmap, Oddmap: ^data_type;
  M, R: mask_array;
procedure Init_mem;
var I,J: integer;
begin
    new(Evenmap); new(Oddmap);      {sets aside space in memory for arrays}
    for J := 0 to down do
      for I := 0 to across do
        begin
          oddmap^ [I,J] := chr(0);      {initializes both arrays}
          evenmap^ [I,J] := chr(0)      {all bytes = 0}
        end
end;     {Init_mem}
procedure Printout;                     {Output to EPSON FX-type printer.}
var  n_lo, n_hi:byte;                   {See listing 2 for Prowriter Printout.}
  i,j : integer;
begin {Printout}
  n_hi:= (across+1) div 256;            {Part of number of graphics bytes coming}
  n_lo:= (across+1) mod 256;            {Rest of number of graphics bytes coming}
  for J := 0 to 39 do
    begin
      write (Lst,chr(27),'Z',chr(n_lo),chr(n_hi));
                                        {Enter graphics mode; give # bytes coming}
      for I := 0 to across do write(Lst, evenmap^ [I,J]); {print even row}
      write(Lst,chr(13));                       {carriage return}
      write(Lst, chr(27),'3',chr(1));           {set linefeed for 1/3 dot down}
      write(Lst, chr(10));                      {do linefeed}
      write (Lst, chr(27),'Z',chr(n_lo),chr(n_hi));     {graphics mode again}
      for I := 0 to across do
        write (Lst, oddmap^ [I,J]);             {print odd row}
      write(Lst, chr(13));                      {carriage return}
      write(Lst,chr(27),'3',chr(22));           {start next line 7 1/3 dots down}
      write(Lst, chr(10));                      {linefeed}
    end
end;  {Printout}
procedure PixelMasks;
var I: integer;
begin
    M[7]:= 1;
    for I:= 6 downto 0 do M[I]:= 2*M[I+1];
    for I:= 0 to 7 do R[I]:= 255 - M[I]
end;  {Pixelmasks}
procedure Change (var Char_byte: char; color, height:integer);
{changes given byte from present value to given value = color}
var  old: integer;
begin
  Old := ord(Char_byte);
  case color of
    1: old:= old OR M[height];          {insert set bit in correct place}
    0: old:= old AND R[height]          {using appropriate pixel masks}
```

(*continued*)

```
    (*******************************************************
    For the Prowriter these last two lines should be replaced by
    1: old:= old OR M[7 - height];
    2: old:= old AND R[7 - height]     {See text for details.}
    *******************************************************)
    end;
  Char_byte:= chr(old)
end;  {Change}
procedure Pset (x,y,color : integer);
{Writes the dot at position (x,y) into memory arrays}
var I,line,height : integer;
begin {Pset}
    Plot(x * 2 div 5, y * 5 div 16, white); {draw dot on screen}
    { *******************************************************
      This multiplies x by the ratio of screen width to printer width,
      multiplies y by the ratio of screen height to printer height.
      For the Prowriter this last line should be replaced by:
      Plot(x div 2, y * 5 div 16, white);
      ******************************************************* }
    color := color mod 2;
    Line := Y div 16;                  {vertical position of pixel consists of a line}
    height := (Y mod 16) div 2;        {number between 0 and down; and a height }
                                       {between 0 and 15, divided into}
                                       {even-odd groups}
    if y mod 2 = 0 then change(evenmap^ [x,line],color,height)
                   else change (oddmap^ [x,line],color,height)
end; {Pset}
```

Listing 2: Printout *procedure for Prowriter.*

```
procedure Printout;                    {for Prowriter}
var wrd : packed array [1..4] of char;
  a,b,i,j,k : integer;
begin {Printout}
  writeln (lpt1,'                          ');     {clear printer buffer}
  writeln (lpt1,'                            ');   { = 50 bytes}
  writeln (lpt1);                      { + carriage return}
  a := across + 1; I := 4;             {a = number of graphics bytes}
  repeat
    b := a mod 10; a := a div 10;      {get next digit (= b)}
    wrd [I] := chr(b + ord( '0'));     {insert as a character in string: wrd}
    I := I - 1
  until I = 0;                         {wrd = digits of across}
  writeln (lpt1, chr(27),'P');         {set pitch for proportional spacing —
                                         the highest horizontal density}
  for J := 0 to down do
      begin
        write (lpt1, chr(27),'S',wrd);          {enter graphics mode}
        for I := 0 to across do
          write (lpt1, evenmap^[I,J]);          {print even rows}
        writeln (lpt1, chr(27),'T','01');       {start next line ½ dot down}
        write (lpt1, chr(27),'S',wrd);          {graphics mode again}
        for I := 0 to across do
          write (lpt1, oddmap^[I,J]);           {print odd rows}
        writeln (lpt1,chr(27),'T','15');        {start next line 7½ dots down}
      end
end;  {Printout}
```

turning on a point involves changing a single bit within a byte. Computers are generally not equipped to do this easily. Remember that each byte controls eight vertical dots, and you want to change *only one* of them. This is most quickly done with bit masks and the logical operations AND and OR. See the text box on bit manipulation, "Bits AND/OR Pieces," on page 225. (Note that in the PixelMasks procedure, the leftmost bit in a byte is called the zeroth bit, while the rightmost bit is the seventh.)

If you want to insert a 1 in the third bit, you use the mask M[height], where height equals 3, with the logical OR operation. The code that inserts this 1 into the byte Old is simply:

```
Old:= Old OR M[height]
```

M[height] is a byte made up of all zeros except for a 1 in bit height. If Old is 01000010 and height is 3, then Old becomes the byte (01000010 OR 00010000) = 01010010.

If you want to insert a 0 into this same byte, you use the mask R[height] together with the logical AND operation:

```
Old:= Old AND R[height]
```

Here, R[height] is a byte made up of 1s except for a 0 in bit height. If Old is 01111101 and height is 7, then Old becomes the byte (01111101 AND 11111110) = 01111100.

Note that you write to printers using Pascal's Write and Writeln procedures, and these procedures expect to be given a character. This is why you should set up your arrays as character arrays and why the last command in the Change procedure converts the byte into a character.

Some Printer Differences

The eight vertically arranged print pins on most printers correspond to the eight bits in a byte. On the Epson FX-80 and many other printers, the high-order bits—those in the left half of the byte—correspond to the upper pins; the low-order, or rightmost, bits correspond to the lower pins. Thus, the byte 10000010 causes the top pin and the next-to-the-bottom pin to

make dots on the paper. On the other hand, for the Prowriter and several other printers, the exact opposite is true—the *leftmost* bit causes the *bottom* pin to fire. Thus, if you want to insert a 1 in the third bit for a Prowriter, you OR with M[7-height] where height equals 3. To avoid confusion we have indicated the corrections necessary to handle the Prowriter properly (see listing 2). If you have a different printer, you should check your manual for the correct pin assignments. (The Prism printer, for example, uses only seven pins.)

Another important difference between printers is in how close you are allowed to print the dots horizontally and vertically. In the Epson quadruple-density graphics mode, available only on the FX, RX, and IBM models, the printer prints 240 dots per inch or 1920 dots across an 8-inch page. Because of restrictions on the size of arrays (64K-byte maximum), the examples in this article draw only 1600 dots. (We can draw more, but at the expense of some vertical rows.) The older Epson MX prints only 960 dots across the page. For the Prowriter, the highest density possible is in proportional mode, where you can get 160 dots per inch or 1280 per line—we use 1250 in our examples.

Each dot on a dot-matrix printer is approximately 1/72 inch in diameter. The Epson FX-80 permits linefeeds of ⅓ dot, which results in a theoretical vertical density of 216 dots per inch. The Prowriter allows ½-dot linefeeds, or a vertical density of 144 dots per inch. In the examples in this article, we use the Epson ⅓-dot linefeeds as if they were ½-dot; this works fine, undoubtedly due to the inherent inaccuracy of paper advance.

Once again, you must consult your printer manual if you have a different printer. The Prism does not seem to support fractional linefeeds at all, while the Mannesmann Tally achieves them by raising or lowering the actual print head ½ dot.

Echoing on the Screen

We now have the complete setup for drawing a dot in "printer" memory. Returning to listing 1, note the call to the procedure Plot. Plot is a Turbo Pascal procedure that draws a dot on the actual screen for each point you draw in memory. However, the scale for the printer is different from the scale for the screen: 1600 by 640 dots for the printer (1250 by 640 dots for the Prowriter) versus 640 by 200 dots (pixels) for the screen. For the Epson FX-80 you rescale by multiplying the column and row, respectively, by 640/1600 (2/5) and 200/640 (5/16). For the Prowriter you multiply by 640/1250 (approximately ½) and 200/640 (5/16). Since real-number multiplication is time-consuming (unless you have an 8087 chip) and since Plot requires integer parameters anyway, you

(*continued*)

Bits AND/OR Pieces

Suppose you have two bytes, each represented as eight binary bits: Byte1 = 10111010 and Byte2 = 00110011. To make calculation simpler later on, let's call the first bit on the left of each byte the zeroth bit; the next is the first, then the second, etc. Thus, the zeroth bit of Byte1 is 1, the first is 0, and the seventh, or rightmost, bit is 0. When you OR Byte1 and Byte2 together, you produce a new byte, Byte3. If either of the corresponding bits, for example the zeroth bits of Byte1 and Byte2, is a 1, then you make the corresponding bit of Byte3 a 1; otherwise, it is a 0. Thus, the zeroth bit of Byte3 is a 1 since Byte1 has a 1 in the zeroth position. The first bit of Byte3 is a 0 since neither Byte1 nor Byte2 has a 1 in that position.

```
Byte1 = 1 0 1 1 1 0 1 0
Byte2 = 0 0 1 1 0 0 1 1
Byte3 = 1 0 1 1 1 0 1 1
Byte3 = Byte1 OR Byte2 =
  10111011.
```

If you perform an AND on the two bytes, the process is similar, except that you put a 1 in Byte3 only if *both* corresponding bits are 1. Let's let Byte4 = Byte1 AND Byte2.

```
Byte1 = 1 0 1 1 1 0 1 0
Byte2 = 0 0 1 1 0 0 1 1
Byte4 = 0 0 1 1 0 0 1 0
Byte4 = Byte1 AND Byte2 =
  00110010
```

Suppose now that you have a byte B = 10011001 and you want to change the second bit from a 0 to a 1. If you have a byte M2 that is all 0s except for a 1 in this second position (i.e., third place from the left), then you can execute

```
B OR M2 = 10011001 OR 00100000
  = 10111001
```

This accomplishes your purpose. You need eight different masks of this type to handle each possible bit position. Note that M2 = 00100000 (binary) = 20 (hexadecimal) = 32 (decimal); also, $32 = 2^{(7-2)}$. All other such masks are powers of 2 also. This explains how M, the array of eight different pixel masks, is constructed in the procedure PixelMasks (see listing 1).

To turn off the fourth bit of B (i.e., change it from a 1 to a 0), you can AND it with a byte R4 that is all 1s except for a 0 in the fourth position (the reverse type of mask from M2):

```
B AND R4 = 10011001 AND 11110111
  = 10010001
```

In this case R4 = 11110111 (binary) = F7 (hexadecimal) = 247 (decimal). The procedure PixelMasks also constructs the array R of eight different reverse masks. The relation between the masks of the two types is easy to see. For example, consider M[3] = 00010000 = 16, and R[3] = 11101111 = 239. Then

```
R[3] = 11101111 = 11111111 - M[3]
  = 255 - M[3]
```

Thus, you get the reverse pixel masks from the normal pixel masks by subtracting the normal ones from 255.

One great advantage of pixel masks is that they are fast. Once created, you can use them over and over without any time-consuming computation. You can use pixel masks in regular screen graphics also; if you use color, you will need several other sets of masks that do two bits at a time, since a choice of one out of four colors requires two bits.

can do this quite neatly using integer multiplication and div:

```
Plot(x * 2 div 5, y * 5 div 16, white)
```

For the Prowriter:

```
Plot(x div 2, y * 5 div 16, white)
```

This is still somewhat wasteful since it draws some dots on top of others, but it is sufficient for this example.

How to Print the Dots

In theory all we have to do is send these bytes to the printer. However, many printers are fussy and don't like to be in graphics mode—in fact, they'll only stay there for one line at a time. Furthermore, each time you invoke graphics mode you have to tell them how many graphics bytes to expect on that line; if you send them more, they start printing regular characters.

Let's do a brief rundown on the Epson FX-80 graphics Printout procedure (see listing 1). Lst is Turbo Pascal's name for the printer. The Epson FX-80 instruction to enter quadruple-density graphics mode is Escape (chr(27)) followed by Z (on the MX, replace Z with L). Then the printer needs to receive the number of graphics bytes it should expect as a sequence of two characters, which are determined as follows:

```
Byte #1 = "n_lo"
  (# of bytes mod 256)
Byte #2 = "n_hi"
  (# of bytes div 256)
```

(This information should be easy to obtain from your printer manual under "Graphics Mode.")

Procedure Printout has two nested loops; the big one controls the printer lines, while the smaller sends out the character bytes within each printer line. Recall that a printer line consists of one even and one odd group of 1600 bytes. For each of these we must, as just mentioned, reenter graphics mode and give the byte count. The command write(Lst, chr(13)) is simply a carriage return.

The only other lines of interest are the paperfeeds. The Epson FX-80 won't do a linefeed of ½ dot but rather works in multiples of ⅓ dot. Since even Epson disclaims any great accuracy for such a tiny linefeed, we tried various combinations such as ⅓ and 7⅔, ⅔ and 7⅓, etc. The best image seemed to result from using ⅓ and 7⅓ (22/3).

Now let's take a look at the Prowriter graphics Printout procedure (see listing 2), since the Prowriter works a little differently. First, you should clear out the 50-byte printer buffer by writing 50 blanks—we've never seen the necessity of this, but it is suggested as a precaution. Next, you should report the number of graphics bytes the printer is to expect (= across + 1) by sending a string whose characters are the *decimal* digits of this number. These are computed by the small loop (from a:= across + 1 through until l = 0;). The rest of the code is the same as the Epson FX-80's except for the different printer instructions (escape sequences).

The Testcurve Program

To demonstrate how these procedures work, listing 3 contains a driver program that sketches the simple parabola $y = x*x$ (see figure 1). The heart of this program is the procedure Plotcurve, which illustrates the scaling and coordinate manipulation necessary to draw "computer pictures." Since the origin is in the upper left-hand corner and the *y*-coordinate is measured downward, you are essentially plotting y = 639 − (x − 25)*(x − 25). x should go from 0 to 50; since the width of the screen is *across* (1599 or 1249), you round *across* to the nearest 50 (width:= across − (across mod 50)) and let l go from 0 to width. The scale factor scaler is width/50 and x equals l/scaler or (l/width)*50. Thus, when l equals 0, x is 0; when l equals width, x is 50. Then you use Pset to graph your points:

```
Pset(l, trunc(639 − (l/scaler − 25)*
  (l/scaler − 25)), 1)
```

Note that you must truncate (trunc) since Pset requires integer parameters.

Connecting the Dots

The procedure Plotcurve draws a curve by computing each point sepa-

(continued)

Listing 3: *Program to test printing procedures. It draws a parabola: y = x*x. Note the* $I *directive to include the routines in* Printpak.pas *(see listing* 1).

```
Program Testcurve;
{$I printpak.pas}        {Include printer procedures listed above.}
var ch: char;
  Procedure Plotcurve;
  var l, width: integer;
      scaler: real;
  begin {Plotcurve}
    width:= across − (across mod 50);
    scaler:= width/50;
    for l:= 0 to width do
      Pset(l, trunc(639−(l/scaler − 25)*(l/scaler − 25)),1)
  end;  {Plotcurve}
begin {Testcurve}
  Init_mem;
  PixelMasks;
  HiRes; HiResColor(7); {draw in 640- by 200-dot mode}
  Plotcurve;
  write('Continue (y/n)? ');
  readln(ch);
  if ch = 'y' then Printout;
  TextMode(BW80)
end.  {Testcurve}
```

rately and then plotting it. Although this sufficed for a simple demonstration, it has two major shortcomings. First, it can skip points. For example, suppose *y* equals 5 when *x* is 1, and *y* equals 10 when *x* is 2. Then there is a vertical gap of four dots between the points (1,5) and (2,10). (This didn't happen on the parabola graphic because *x* went from 0 to 50 in 1599 steps, so each step represented an *x* change of about 0.03. Thus, even at the steepest part of the curve, *y*

(continued)

Listing 4: *Bresenham's line-drawing algorithm. (The Pascal implementation is courtesy of Professor Richard Rasala of Northeastern University.)*

```
Procedure Pixel_Line(x1,y1,x2,y2:integer);
var x, y, z, a, b, dx, dy, d, deltap, deltaq: integer;
begin
  dx:= abs(x2 - x1);
  dy:= abs(y2 - y1);
  If dy <= dx then {Slope <= 1}
    begin
    x:= x1; {initialize x}
    y:= y1; {initialize y}
    z:= x2; {set sentinel in x-direction}
    {Now set x-increment}
    If x1 <= x2
      then a:= 1 {x increases}
      else a:= -1; {x decreases}
    {Now set y-increment}
    If y1 <= y2
      then b:= 1      {y increases}
      else b:= -1;    {y decreases}
    {Initialize decision function and its deltas}
    deltap:= dy + dy;
    d     := deltap - dx;
    deltaq:= d - dx;
    {Locate and plot points}
    Pset(x,y,1);     {First point}
    while x <> z do begin
        x:= x + a;
        if d < 0
          then d:= d+ deltap
          else begin
            y:= y + b;
            d:= d + deltaq
            end;    {else}
        Pset(x,y,1);
        end   {while}
  end    {Case: if dy <= dx}
else     {dx <= dy so view x as a function of y}
  begin
    y:= y1;   {initialize y}
    x:= x1;   {initialize x}
    z:= y2;   {set sentinel in y-direction}
    {Now set y-increment}
    If y1 <= y2
      then a:= 1      {y increases}
      else a:= -1;    {y decreases}
    {Now set x-increment}
    If x1 <= x2
      then b:= 1      {x increases}
      else b:= -1;    {x decreases}
    {Initialize decision function and its deltas}
```

(continued)

```
      deltap:= dx + dx;
      d     := deltap - dy;
      deltaq:= d - dy;
      {Locate and plot points}
      Pset(x,y,1);      {First point}
      while y <> z do begin
        y:= y + a;
        if d < 0
          then d:= d+ deltap
          else begin
            x:= x + b;
            d:= d + deltaq
            end;      {else}
        Pset(x,y,1);
        end   {while}
      end   {else}
end;        {Pixel_line}
```

changes only about 1.5 dots per change in *x*—hardly visible at over 200 dots per inch.)

Second, this point-by-point calculation takes time. Even when the curve is smooth or nearly straight, every point must be calculated. For curves from simple functions this doesn't produce too much overhead, but for complicated mathematical equations or for curves produced by rotating images, this "overcalculation" is unacceptably slow.

The solution to both of these problems is to compute fewer points and to join the points computed with simple, easy-to-calculate curves. For most purposes these simple curves can be taken to be straight lines. If you only compute every fifth point and you connect the points by lines, there is a considerable time savings if point computations are reasonably complex and the line-drawing algorithm is fast. Furthermore, this solves the problem of gaps, since, in the example above, the points (1,5) and (2,10) would be joined by a small line segment "filling in" the missing four points.

The problem, then, is finding a fast line-drawing algorithm. Trying to find the equation of the line joining two points and then plotting it requires a considerable amount of real-number (decimal) arithmetic. This kind of arithmetic, especially multiplication and division, is quite slow in comparison with whole-number manipulation. Furthermore, since the coordinates of points on the screen (or printer page) are always integers—column and row numbers—you would naturally hope for a whole-number algorithm. Fortunately, there is one, called the Bresenham Line Algorithm (named for its inventor, J. E. Bresenham). It not only computes the points on the line connecting any two screen points, using whole-number arithmetic, but it accomplishes this feat *without using either multiplication or division!* Listing 4 contains a Pascal implementation of it. The procedure call is Pixel_line(x1,y1,x2,y2,color) where x1,y1 and x2,y2 are the endpoints of the line. For an easy-to-read description of the theory behind Pixel_line, see *Fundamentals of Interactive Computer Graphics* by James D. Foley and Andries Van Dam (Addison-Wesley, 1982).

Sometimes, when speed is even more important and points are very time-consuming to compute, you must cut down radically on the number of points calculated. Joining the points by straight lines will usually produce a figure that is too polygonal in appearance. In figure 1, the points are joined by curved pieces called *splines*, for which there are now very fast computational algorithms. There is some discussion of splines in Foley and Van Dam's book, but the

(continued)

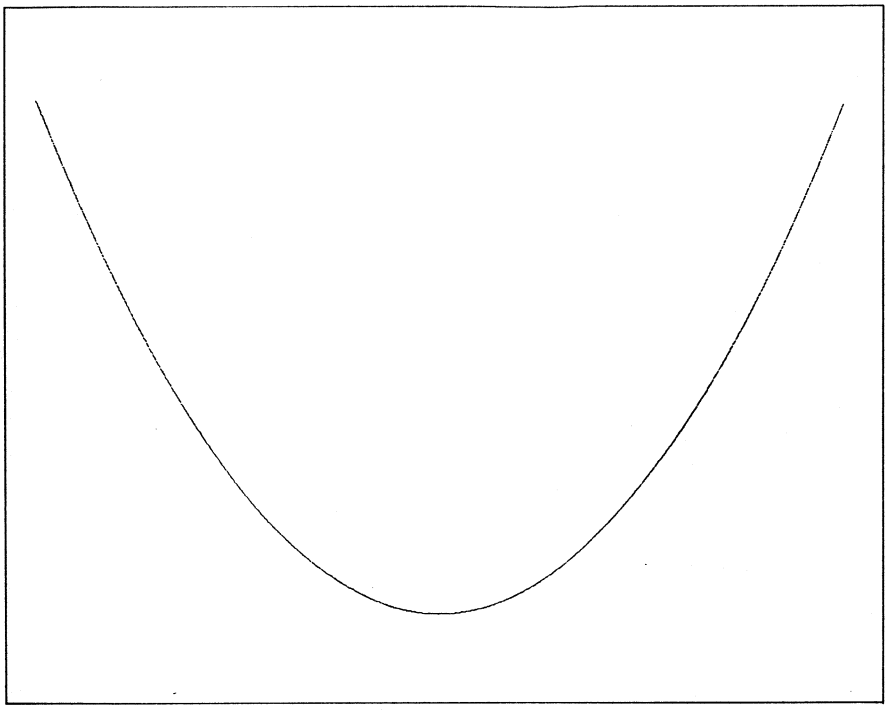


Figure 1: *High-resolution plot of a parabola (y = x*x) created on an Epson FX-80 printer.*

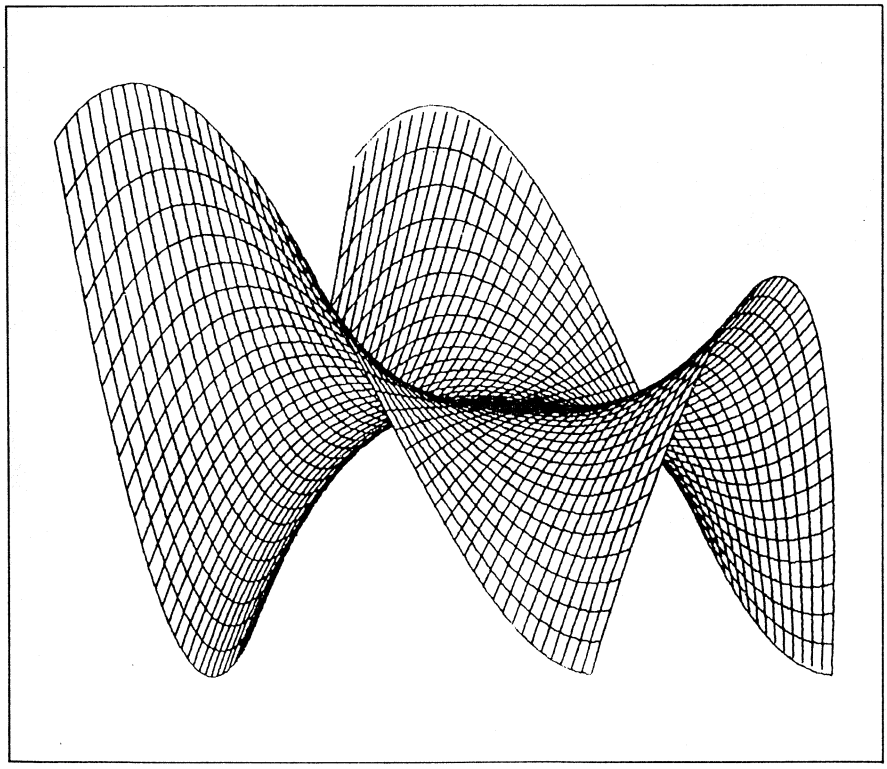


Figure 2: *High-resolution plot of the surface $z = x^3 - 3xy^2$ (1600 by 640 dots prepared using the Epson FX-80 printer and the Bridge Software Math Utilities).*

There are clever ways of getting even more speed out of the line drawing—especially for lines of small slope—by exploiting block moves of bytes.

most efficient algorithms are to be found in the current technical computer science journals.

Further Applications and Extensions

Armed with procedures for drawing points and lines on the screen and on the printer, you can implement procedures for making very complex high-resolution pictures. It is possible, given enough memory, to set aside more pairs of arrays to increase further the image size you can print. This is the reason to use dynamic variables, the ones with the "^".

It is also possible to print your picture sideways, but this requires a restructuring of the procedure Change so that it addresses the points correctly.

Finally, you can use pixel masks to draw points on the graphics screen as well as the printer. The point and line-drawing procedures included in BASIC and Turbo Pascal, for example, are implemented by combining color and monochrome pixel masks with some version of Bresenham line drawing. There are clever ways of getting even more speed out of the line drawing—especially for lines of small slope—by exploiting *block* moves of bytes.

Figure 2 shows a surface plotted by an Epson FX-80 printer with a resolution of 1600 by 640 dots. It indicates the complexity of drawing possible with this method of printer addressing. ■

```
      SUBROUTINE PRSPEC (Z, IROW, ICOL, NROW, NCOL, IV, A, B, G, PSIZE)
      DIMENSION Z(NROW,NCOL), IV(NROW,NCOL)
      COMMON/PRSHFT/SHIFT
      DATA   CONV/0.0174533/
      SHIFT = 1.5
      D = -1.0
C**   A=DIST. OF VIEW POINT TO PERSPECTIVE PLANE, B AND G ARE
C**   EULERS ANGLES OF VIEW POINT ORIENTATION, D= DIST. OF
C**  VIEWING PLANE FROM VIEW POINT (USUALLY NEGATIVE)
      CX = A*SIN(B*CONV)*COS(G*CONV)
      CY = A*SIN(B*CONV)*SIN(G*CONV)
      CZ = A*COS(B*CONV)
      CX = -CX
C**   CX,CY,CZ ARE RECTANGULAR CO-ORDINATES OF VIEW POINT.
C**   PSIZE GIVES SIZE OF SQUARE PLOT IS TO FIT IN.
      CALL PERPKS (CX, CY, CZ, D, Z, IROW, ICOL, IV, PSIZE, NROW, NCOL)
C**    PERPKS COMPUTES TRANSFORMATION CONSTANTS AND SCALES PLOT
      RETURN
      END
      SUBROUTINE PERPKS (CX, CY, CZ, D, Z, IR, IC, IV, PSIZE, NROW, NCOL
     1)
      DIMENSION Z(NROW,NCOL), IV(NROW,NCOL), XYZ(8,2)
      INTEGER BOBSK
C**    PARAMETERS
C**    CX,CY,CZ = COORDINATES OF VANISHING POINT.
C**    D = DISTANCE FROM VANISHING POINT TO PROJECTION PLANE.
C**    Z = GRID OF SURFACE VALUES, IR,IC = ACTUAL DIMENSIONS
C**    IV = SCRATCH ARRAY USED IN HIDDEN LINE ELIMINATION.
C**    PSIZE = WIDTH IN INCHES OF PLOT
      COMMON /NORM/ SC, TC, PPP
      COMMON /CUT/ XCUT, YCUT
      COMMON /ISOMC/ XB, YB
      COMMON /CONS/ AX, AY, AZ, ALPHA, BETA, GAMMA, DD, GASN, BASN, QX,
     1QY, QZ
      COMMON /XCON/ ZMN, ZSCL
      COMMON / RESCAL / BOBSK, NCALL
C
C   SETTING BOBSK TO 0 CAUSES PERSPEC TO RESCALE ONLY IF NCALL=0
C   NCALL IS INITIALLY ZERO
C
      BOBSK = -1
      NCALL = 0
      DIST = SQRT(CX**2+CY**2+CZ**2)
C**    COMPUTE PERSPECTIVE TRANSFORM INFO
      ALPHA = CX/DIST
      BETA = CY/DIST
      GAMMA = CZ/DIST
      GASN = SIN(ACOS(GAMMA))
      BASN = SIN(ACOS(BETA))
      AX = CX
      AY = CY
      AZ = CZ
      DD = D
      QX = CX+D*ALPHA
      QY = CY+D*BETA
      QZ = CZ+D*GAMMA
C**    SCALE VALUES FOR PLOT
      IF ( BOBSK .EQ. 0 .AND. NCALL .NE. 0 ) GO TO 14
      ZMX = Z(1,1)
      ZMN = Z(1,1)
      DO 5 I = 1, IR
      DO 5 J = 1, IC
      ZMX = AMAX1(ZMX, Z(I, J))
 5    ZMN = AMIN1(ZMN, Z(I, J))
      IF (ZMX .NE. ZMN) GO TO 10
      RETURN
 10   CONTINUE
      ZSCL = AMAX0(IR, IC)/(ZMX-ZMN)
 14   DO 15 I = 1, IR
      DO 15 J = 1, IC
 15   Z(I, J) = (Z(I, J)-ZMN)*ZSCL
      NCALL = NCALL + 1
      XDIM = AMAX0(IR, IC)
      XCUT = 0.
      YCUT = 0.
      CALL ISOM (0, 0, 0)
      XYZ(1, 1) = XB
      XYZ(1, 2) = YB
      CALL ISOM (XDIM, 0, 0)
      XYZ(2, 1) = XB
      XYZ(2, 2) = YB
      CALL ISOM (0, XDIM, 0)
      XYZ(3, 1) = XB
      XYZ(3, 2) = YB
      CALL ISOM (0, 0, XDIM)
```

```
      XYZ(4, 1) = XB
      XYZ(4, 2) = YB
      CALL ISOM (0, XDIM, XDIM)
      XYZ(5, 1) = XB
      XYZ(5, 2) = YB
      CALL ISOM (XDIM, XDIM, 0)
      XYZ(6, 1) = XB
      XYZ(6, 2) = YB
      CALL ISOM (XDIM, 0, XDIM)
      XYZ(7, 1) = XB
      XYZ(7, 2) = YB
      CALL ISOM (XDIM, XDIM, XDIM)
      XYZ(8, 1) = XB
      XYZ(8, 2) = YB
      XMX = XYZ(1, 1)
      XMN = XYZ(1, 1)
      YMX = XYZ(1, 2)
      YMN = XYZ(1, 2)
      DO 20 I = 1, 8
      XMX = AMAX1(XMX, XYZ(I, 1))
      XMN = AMIN1(XMN, XYZ(I, 1))
      YMX = AMAX1(YMX, XYZ(I, 2))
      YMN = AMIN1(YMN, XYZ(I, 2))
 20   CONTINUE
      DX = XMX-XMN
      DY = YMX-YMN
      IF (DY .EQ. 0.) DY = 1.0E-11
      IF (DX .EQ. 0.) DX = 1.0E-11
      IF (DY .GT. DX) GO TO 25
      SC = PSIZE/DX
      GO TO 30
 25   SC = PSIZE/DY
 30   TC = SC
      XCUT = XMN
      YCUT = YMN
C**   INITIAL VALUES FOR HIDDEN AND NON-HIDDEN LINES
      DO 35 I = 1, IR
      DO 35 J = 1, IC
 35   IV(I, J) = IVIS(FLOAT(J), FLOAT(I), Z(I, J), Z, IR, IC, NROW, NCOL
     1)
      CALL MAPIP (Z, IR, IC, IV, NROW, NCOL)
C**    MAPIP DRAWS PERSPECTIVE VIEW
      RETURN
      END
      SUBROUTINE MAPIP (Z, IR, IC, IPHI, NROW, NCOL)
      DIMENSION Z(NROW,NCOL), IPHI(NROW,NCOL)
C**    MAPIP DRAWS PERSPECTIVE VIEW OF SURFACE
C**    Z = GRID OF SURFACE VALUES. IR,IC = LOGICAL DIMENSIONS
C**    IPHI = AUXILIARY ARRAY.
      COMMON /ISOMC/ XXX, YYY
      COMMON /NORM/ XNORM, YNORM, PSZE
      IOLD = IPHI(1,1)
      XOLD = 1.
      YOLD = 1.
      ZOLD = Z(1,1)
      CALL ISOM (XOLD, YOLD, ZOLD)
      CALL PLOT (XXX*XNORM, YYY*YNORM, 3)
C**  PLOTS IN CONTINUOUS LINES
C**    PLOT ROWS
      DO 5 I = 1, IR
      ICODE = MOD(I, 2)
      DO 5 JJ = 1, IC
      J = JJ
      IF (ICODE .EQ. 0) J = IC-JJ+1
      INEW = IPHI(I, J)
      XNEW = J
      YNEW = I
      ZNEW = Z(I, J)
      CALL PRAW (INEW, XNEW, YNEW, ZNEW, IOLD, XOLD, YOLD, ZOLD, Z, IR,
     1IC, NROW, NCOL)
      IOLD = INEW
      XOLD = XNEW
      YOLD = YNEW
      ZOLD = ZNEW
 5    CONTINUE
      IFLAG = 0
      IF (J .EQ. IC) IFLAG = 1
C**    PLOT COLUMNS
      DO 10 JJ = 1, IC
      J = JJ
      IF (IFLAG .EQ. 1) J = IC-JJ+1
      ICODE = MOD(JJ, 2)
      DO 10 II = 1, IR
      I = II
      IF (ICODE .EQ. 1) I = IR-II+1
```

```
      INEW = IPHI(I, J)
      XNEW = J
      YNEW = I
      ZNEW = Z(I, J)
      CALL PRAW (INEW, XNEW, YNEW, ZNEW, IOLD, XOLD, YOLD, ZOLD, Z, IR,
     1IC, NROW, NCOL)
      IOLD = INEW
      XOLD = XNEW
      YOLD = YNEW
      ZOLD = ZNEW
 10   CONTINUE
      RETURN
      END
      SUBROUTINE PRAW (INEW, XNEW, YNEW, ZNEW, IOLD, XOLD, YOLD, ZOLD,Z
     1, IR, IC, NROW, NCOL)
      DIMENSION Z(NROW,NCOL)
      COMMON /NORM/ XNORM, YNORM, SIZE
      COMMON /BINVC/ XVIS, YVIS, ZVIS
      COMMON /ISOMC/ XBAR, YBAR
      IF (INEW .NE. IOLD) GO TO 10
C**   BOTH HIDDEN
      IF (INEW .EQ. 0) RETURN
      CALL ISOM (XNEW, YNEW, ZNEW)
C**  BOTH VISIBLE
5     CALL PLOT (XBAR*XNORM, YBAR*YNORM, 2)
      RETURN
 10   IF (INEW .NE. 0) GO TO 15
C**  OLD VISIBLE, NEW HIDDEN
      CALL BINVIS (XOLD, YOLD, XNEW, YNEW, IOLD, Z, IR, IC, NROW, NCOL)
      CALL ISOM (XVIS, YVIS, ZVIS)
      GO TO 5
 15   IF (IOLD .NE. 0) GO TO 20
C**   OLD HIDDEN, NEW VISIBLE
      CALL BINVIS (XNEW, YNEW, XOLD, YOLD, INEW, Z, IR, IC, NROW, NCOL)
      CALL ISOM (XVIS, YVIS, ZVIS)
      CALL PLOT (XBAR*XNORM, YBAR*YNORM, 3)
      CALL ISOM (XNEW, YNEW, ZNEW)
      GO TO 5
 20   CALL BINVIS (XOLD, YOLD, XNEW, YNEW, IOLD, Z, IR, IC, NROW, NCOL)
C**   HIDDEN PORTION BETWEEN LINES
      CALL ISOM (XVIS, YVIS, ZVIS)
      CALL PLOT (XBAR*XNORM, YBAR*YNORM, 2)
      CALL BINVIS (XNEW, YNEW, XOLD, YOLD, INEW, Z, IR, IC, NROW, NCOL)
      CALL ISOM (XVIS, YVIS, ZVIS)
      CALL PLOT (XBAR*XNORM, YBAR*YNORM, 3)
      CALL ISOM (XNEW, YNEW, ZNEW)
      GO TO 5
      END
      SUBROUTINE BINVIS (X1, Y1, X2, Y2, LVIS, Z, IR, IC, NROW, NCOL)
      DIMENSION Z(NROW,NCOL)
C* RETURNS (XV,YV), INTERIOR POINT THAT AGREES WITH IVIS AND
C* IS LESS THAN .1 FROM POINT WHICH DOES NOT AGREE, X1, Y1 AGREE
C* WITH IVIS, X1. Y2 DO NOT, EXAMINES OPEN INTERVAL BETWEEN.
      COMMON /BINVC/ XV, YV, ZV
      XV = X1
      YV = Y1
      XN = X2
      YN = Y2
 5    XMID = (XV+XN)/2.
      YMID = (YV+YN)/2.
      ZV = ZSTAR(XMID, YMID, Z, IR, IC, NROW, NCOL)
      IF (IVIS(XMID, YMID, ZV, Z, IR, IC, NROW, NCOL) .NE. LVIS) GO TO 1
     10
      XV = XMID
      YV = YMID
      GO TO 15
 10   XN = XMID
      YN = YMID
C**   HOW ACCURATE MAP IS TO BE
 15   IF ((XV-XN)**2+(YV-YN)**2 .GT. .0001) GO TO 5
      RETURN
      END
      FUNCTION ZSTAR (XB, YB, Z, IR, IC, NROW, NCOL)
C*RETURNS APPROXIMATION TO Z(XB,YB)
      DIMENSION Z(NROW,NCOL)
      I = XB
      J = YB
C*QUESTIONABLE APPROXIMATION AT EDGES
      IF (I .EQ. IC) I = I-1
      I1 = I+1
      IF (J .EQ. IR) J = J-1
      J1 = J+1
      Z1 = Z(J1, I)
      Z2 = Z(J, I1)
      IF (YB-J .GT. I1-XB) GO TO 5
```

```
      Z3 = Z(J, I)
      A = Z2-Z3
      B = Z1-Z3
      C = Z3-A*I-B*J
      GO TO 10
 5    Z3 = Z(J1, I1)
      A = Z3-Z1
      B = Z3-Z2
      C = Z1-A*I-B*J1
 10   ZSTAR = A*XB+B*YB+C
      RETURN
      END
      FUNCTION IVIS (X, Y, E, Z, IR, IC, NROW, NCOL)
C**   DETERMINES IF POINT (X,Y,E) IS VISIBLE FROM POINT (CX,CY,CZ)
      COMMON /CONS/ CX, CY, CZ, ISC(9)
      DIMENSION Z(NROW,NCOL)
      DX = X-CX
      DY = Y-CY
      DZ = E-CZ
      D = DX*DX+DY*DY
      IX = X
      IY = Y
      ICODE = 0
      IF (DY .EQ. 0.) GO TO 15
      DO 10 IYB = 1, IR
      IHIT = 1
      YB = IYB
      XB = (YB-CY)/DY*DX+CX
      ZB = (YB-CY)/DY*DZ+CZ
      DB = (XB-CX)**2+(YB-CY)**2
      IF (D .LE. DB+2.0E-8) GO TO 10
      IXB = XB
      IXB1 = IXB+1
      GO TO (2,3),IHIT
2     CONTINUE
      IF (IXB .LT. 1 .OR. IXB1 .GT. IC) GO TO 10
      IHIT = 2
      GO TO 4
3     IF (IXB .LT. 1 .OR. IXB1 .GT. IC) GO TO 15
4     CONTINUE
      ZS = (XB-IXB)*(Z(IYB, IXB1)-Z(IYB, IXB))+Z(IYB, IXB)
      G = ZB-ZS
      IF (ICODE .NE. 0) GO TO 5
      GL= G
      ICODE = 1
5     IF (GL*G .LT. 0) GO TO 40
      IF( G .NE. 0) GL=G
 10   CONTINUE
 15   IF (DX .EQ. 0.) GO TO 30
      DO 25 IXB = 1, IC
      IHIT = 1
      XB = IXB
      YB = (XB-CX)/DX*DY+CY
      ZB = (XB-CX)/DX*DZ+CZ
      DB = (XB-CX)**2+(YB-CY)**2
      IF (D .LE. DB+2.0E-8) GO TO 25
      IYB = YB
      IYB1 = IYB+1
      GO TO (16,17),IHIT
16    CONTINUE
      IF (IYB .LT. 1 .OR. IYB1 .GT. IR) GO TO 25
      IHIT = 2
      GO TO 18
17    IF (IYB .LT. 1 .OR. IYB1 .GT. IR) GO TO 30
18    CONTINUE
      ZS = (YB-IYB)*(Z(IYB1, IXB)-Z(IYB, IXB))+Z(IYB, IXB)
      G = ZB-ZS
      IF (ICODE .NE. 0) GO TO 20
      GL = G
      ICODE = 1
20    IF (GL*G .LT. 0) GO TO 40
      IF (G .NE. 0) GL=G
 25   CONTINUE
 30   IF (ICODE .NE. 0) GO TO 35
      GL = 1.
      IF (CZ .LT. 0.) GL = -1.
 35   IFLG = 1
      IF (GL .LT. 0.) IFLG = -1
      IVIS = IFLG
      RETURN
 40   IVIS = 0
      RETURN
      END
      SUBROUTINE ISOM (X, YYY, Z)
C**   DOES PERSPECTIVE TRANSFORMATION, GIVEN (X,YYY,Z) YIELDS (XB,YB)
```

```
C**   XD AND YD ARE SCALE FACTORS SUBTRACTED
      COMMON /ISOMC/ XB, YB
      COMMON /CONS/ CX, CY, CZ, ALPHA, BETA, GAMMA, D, GAMASN, BETASN,
     1QX, QY, QZ
      COMMON /CUT/ XD, YD
      REAL NU, K
      Y = YYY
      K = D/((X-CX)*ALPHA+(Y-CY)*BETA+(Z-CZ)*GAMMA)
      ZI = CX+K*(X-CX)
      NU = CY+K*(Y-CY)
      ZETA = CZ+K*(Z-CZ)
      IF (GAMASN .EQ. 0.) GO TO 5
      XB = ((ZI-QX)*BETA-(NU-QY)*ALPHA)/GAMASN
      YB = (ZETA-QZ)/GAMASN
      XB = XB-XD
      YB = YB-YD
      RETURN
 5    IF (BETASN .EQ. 0.) GO TO 10
      XB = ((-ZI+QX)*GAMMA+(ZETA-QZ)*ALPHA)/BETASN
      YB = (NU-QY)/BETASN
      XB = XB-XD
      YB = YB-YD
      RETURN
 10   XB = X
      YB = Y
      XB = XB-XD
      YB = YB-YD
      RETURN
      END
      SUBROUTINE CNTR3D (Z, IR, IC, NROW, NCOL, NCONT)
C
C  THIS ROUTINE CONTROLS THE COMPUTATION OF PERSPECTIVE CONTOURS.
C  A CONTOUR LINE ON A COMPLICATED SURFACE CAN CONSIST OF SEVERAL
C  CONTINUOUS SEGMENTS, AND DEEP IN THE BOWELS OF THIS CODE A NONTRIVIAL
C  AMOUNT OF SORTING AND REINDEXING IS DONE TO DESCRIBE A GIVEN CONTOUR
C  BY THE MINIMUM NUMBER OF CONTINUOUS SEGMENTS.  THE CURRENT
C  DIMENSIONS ALLOW 199 SEGMENTS AT A GIVEN INTERMEDIATE STAGE OF COMPUT
C  TION.  IF YOUR PLOT EXCEEDS THIS NUMBER INFINITE LOOPING, TRUNCATED
C  LINES AND MUCH OTHER GRIEF CAN BEFALL YOU.  BEWARE.
C
      DIMENSION Z(NROW,NCOL)
      DIMENSION CONVLS(50)
      CALL BOUNDS (Z, CONVLS, NROW, NCOL, IR, IC, NCONT, ZMX, ZMN)
      N1 = NCONT
      DO 5 I = 1, N1
      CALL CONLIN (Z, IR, IC, NROW, NCOL, CONVLS(I))
      CALL PLTCRV (Z, IR, IC, CONVLS(I), NROW, NCOL)
 5    CONTINUE
      RETURN
      END
      SUBROUTINE BOUNDS (Z, CONVLS, IZ, JZ, IROW, ICOL, NUMBRC, ZMAX,
     1ZMIN)
C****BOUNDS GIVEN ARRAY Z AND ITS INDICES IROW,ICOL AND THE NUMBER OF
C****CONTOUR LINES TO BE PLOTTED,NUMBRC, COMPUTES ZMAX AND ZMIN THEN
C****DETERMINES THE CONTOUR VALUE FOR EACH LINE.
      DIMENSION Z(IZ,JZ)
      DIMENSION CONVLS(50)
      ZMAX = Z(1,1)
      ZMIN = Z(1,1)
      DO 5 I = 1, IROW
      DO 5 J = 1, ICOL
      TEMPZ = Z(I, J)
      ZMAX = AMAX1(ZMAX, TEMPZ)
      ZMIN = AMIN1(ZMIN, TEMPZ)
 5    CONTINUE
      DELTA = (ZMAX-ZMIN)/(NUMBRC+1.)
      DO 10 K = 1, NUMBRC
      CONVLS(K) = ZMIN+K*DELTA
 10   CONTINUE
      RETURN
      END
      SUBROUTINE CONLIN (Z, JZ, IZ, IMX, JMX, CONVAL)
      DIMENSION Z(IMX,JMX)
      COMMON /CNTR/ X1, X2, X3, X4, Y1, Y2, Y3, Y4, Z1, Z2, Z3, Z4
      CALL SRTSEG (1, GRID1,GRID2,GRID3,GRID4)
      IZ1 = IZ-1
      JZ1 = JZ-1
      DO 5 J = 1, JZ1
      YP = J
      DO 5 I = 1, IZ1
      XP = I
      Z1 = Z(J, I)
      Z2 = Z(J+1, I)
      Z3 = Z(J+1, I+1)
      Z4 = Z(J, I+1)
```

```
C**   FINDS END POINTS OF LINE ACROSS GRID SQUARES
      CALL ENDSEG (ISEG, CONVAL)
      IF (ISEG .EQ. 0) GO TO 5
      CALL SRTSEG (2, X1+XP, Y1+YP, X2+XP, Y2+YP)
      IF (ISEG .EQ. 1) GO TO 5
      CALL SRTSEG (2, X3+XP, Y3+YP, X4+XP, Y4+YP)
 5    CONTINUE
C**   GROUPS ALL SEGMENTS INTO STRINGS
      CALL SRTSEG (3, GRID1,GRID2,GRID3,GRID4)
      RETURN
      END
      SUBROUTINE PLTCRV (Z, IR, IC, CONVL, NROW, NCOL)
      COMMON /CTCOM1/ X(600),Y(600),NEXT(600), IBEG(200), IEND(200), IOM
     1IT(200), NCOUNT(200), KTOTAL
      COMMON /NORM/ XNORM, YNORM, ZSCRAT
      COMMON /XCON/ ZMN, ZSCL
      COMMON /ISOMC/ XB, YB
      DIMENSION Z(NROW,NCOL)
      IF (KTOTAL .EQ. 0) RETURN
      DO 10 K = 1, KTOTAL
      IST = IOMIT(K)
      IB = IBEG(IST)
      XOLD = X(IB)
      YOLD = Y(IB)
      ZOLD = CONVL
      IOLD = IVIS(XOLD, YOLD, ZOLD, Z, IR, IC, NROW, NCOL)
      CALL ISOM (XOLD, YOLD, ZOLD)
      CALL PLOT (XB*XNORM, YB*YNORM, 3)
 5    CONTINUE
      IB = NEXT(IB)
      IF (IB .EQ. 0) GO TO 10
      XNEW = X(IB)
      YNEW = Y(IB)
      ZNEW = ZOLD
      INEW = IVIS(XNEW, YNEW, ZNEW, Z, IR, IC, NROW, NCOL)
      CALL PRAW (INEW, XNEW, YNEW, ZNEW, IOLD, XOLD, YOLD, ZOLD, Z, IR,
     1IC, NROW, NCOL)
      IOLD = INEW
      XOLD = XNEW
      YOLD = YNEW
      ZOLD = ZNEW
      GO TO 5
 10   CONTINUE
      RETURN
      END
      SUBROUTINE ENDSEG (ISEG, CONVAL)
C****SUBROUTINE TO FIND THE END POINTS OF A CONTOUR SEGMENT ACROSS
C****A GRID SQUARE
      DIMENSION X(4), Y(4)
      EQUIVALENCE (X(1), XA1), (X(2), XA2), (X(3), XA3), (X(4), XA4)
      EQUIVALENCE (Y(1), YA1), (Y(2), YA2), (Y(3), YA3), (Y(4), YA4)
      COMMON /CNTR/ X1, X2, X3, X4, Y1, Y2, Y3, Y4, Z1, Z2, Z3, Z4
      ISEG = 0
      IS1 = 0
      IS2 = 0
      IS3 = 0
      IS4 = 0
      IF (Z1-CONVAL) 5,10,10
 5    IF (Z2-CONVAL) 20,15,15
 10   IF (Z2-CONVAL) 15,20,20
 15   IS1 = 1
 20   CONTINUE
      IF (Z2-CONVAL) 25,30,30
 25   IF (Z3-CONVAL) 40,35,35
 30   IF (Z3-CONVAL) 35,40,40
 35   IS2 = 1
 40   CONTINUE
      IF (Z3-CONVAL) 45,50,50
 45   IF (Z4-CONVAL) 60,55,55
 50   IF (Z4-CONVAL) 55,60,60
 55   IS3 = 1
 60   CONTINUE
      IF (Z4-CONVAL) 65,70,70
 65   IF (Z1-CONVAL) 80,75,75
 70   IF (Z1-CONVAL) 75,80,80
 75   IS4 = 1
 80   IF (IS1+IS2+IS3+IS4-2) 155,85,85
 85   I = 0
      IF (IS1) 95,95,90
 90   I = I+1
      X(I) = 0.
      Y(I) = (Z1-CONVAL)/(Z1-Z2)
 95   IF (IS2) 105,105,100
 100  I = I+1
      X(I) = (Z2-CONVAL)/(Z2-Z3)
```

```
      Y(I) = 1.
  105 IF (IS3) 115,115,110
  110 I = I+1
      X(I) = 1.
      Y(I) = (Z4-CONVAL)/(Z4-Z3)
  115 IF (IS4) 125,125,120
  120 I = I+1
      X(I) = (Z1-CONVAL)/(Z1-Z4)
      Y(I) = 0.
  125 IF (I-4) 130,135,135
  130 ISEG = 1
      GO TO 150
  135 ISEG = 2
      D1 = (XA1-XA2)
      DY = (YA1-YA2)
      D1 = D1*D1+DY*DY
      D2 = (XA2-XA3)
      DY = (YA2-YA3)
      D2 = D2*D2+DY*DY
      D3 = (XA3-XA4)
      DY = (YA3-YA4)
      D3 = D3*D3+DY*DY
      D4 = (XA4-XA1)
      DY = (YA4-YA1)
      D4 = D4*D4+DY*DY
      IF (D1-D2) 140,140,160
  140 IF (D1-D3) 145,145,165
  145 IF (D1-D4) 150,150,170
  150 X1 = XA1
      X2 = XA2
      X3 = XA3
      X4 = XA4
      Y1 = YA1
      Y2 = YA2
      Y3 = YA3
      Y4 = YA4
  155 RETURN
  160 IF (D2-D3) 170,170,165
  165 IF (D3-D4) 150,150,170
  170 X1 = XA2
      X2 = XA3
      X3 = XA1
      X4 = XA4
      Y1 = YA2
      Y2 = YA3
      Y3 = YA1
      Y4 = YA4
      RETURN
      END
      SUBROUTINE SRTSEG (NTASK, SX1, SY1, SX2, SY2)
C****SEGMENT-STORE ROUTINE WITH DYNAMIC STORAGE ALLOCATION
      DIMENSION X(600), Y(600), NEXT(600)
      DIMENSION IBEG(200), IEND(200), IOMIT(200), NCOUNT(200)
      COMMON /CTCOM1/ X, Y, NEXT, IBEG, IEND, IOMIT, NCOUNT, KTOTAL
C**   INITIALIZE LISTS
      GO TO (5,25,160), NTASK
   5  KMAX = 15
      DO 10 J = 1, 600
  10  NEXT(J) = 0
      DO 15 J = 1, 200
      NCOUNT(J) = 0
  15  IOMIT(J) = 1
      KTOTAL = 0
      KMIN = 1
      KSTOR = 0
  20  RETURN
C**   STORE END POINTS OF CONTOUR LINES ACROSS GRID SQUZRE
  25  IF (KSTOR) 100,100,30
  30  DO 80 I = KMIN, KTOTAL
      INDEX = IBEG(I)
      TMPX = X(INDEX)
      TMPY = Y(INDEX)
      K = NEQL(SX1, SY1, TMPX, TMPY)
      GO TO (35,45), K
  35  X(KSTOR) = SX2
      Y(KSTOR) = SY2
  40  NCOUNT(I) = NCOUNT(I)+1
      NEXT(KSTOR) = INDEX
      IBEG(I) = KSTOR
      KSTOR = KSTOR+1
      IF (KSTOR-KMAX) 20,20,105
  45  K = NEQL(SX2, SY2, TMPX, TMPY)
      GO TO (50,55), K
  50  X(KSTOR) = SX1
      Y(KSTOR) = SY1
```

```
      GO TO 40
 55   INDEX = IEND(I)
      TMPX = X(INDEX)
      TMPY = Y(INDEX)
      K = NEQL(SX1, SY1, TMPX, TMPY)
      GO TO (60,70), K
 60   X(KSTOR) = SX2
      Y(KSTOR) = SY2
 65   NCOUNT(I) = NCOUNT(I)+1
      NEXT(INDEX) = KSTOR
      IEND(I) = KSTOR
      KSTOR = KSTOR+1
      IF (KSTOR-KMAX) 20,20,105
 70   K = NEQL(SX2, SY2, TMPX, TMPY)
      GO TO (75,80), K
 75   X(KSTOR) = SX1
      Y(KSTOR) = SY1
      GO TO 65
 80   CONTINUE
 85   X(KSTOR) = SX1
      Y(KSTOR) = SY1
      NNEXT = KSTOR+1
      NEXT(KSTOR) = NNEXT
      X(NNEXT) = SX2
      Y(NNEXT) = SY2
      KTOTAL = KTOTAL+1
      IF (KTOTAL-199) 95,95,90
 90   KTOTAL = 199
 95   IBEG(KTOTAL) = KSTOR
      IEND(KTOTAL) = NNEXT
      KSTOR = KSTOR+2
      NCOUNT(KTOTAL) = 1
      IF (KSTOR-KMAX) 20,20,105
 100  KSTOR = 1
      GO TO 85
C****ROUTINE TO DIMINISH SEGMENT CHECK-LIST
 105  KMAX = KMAX+15
      SXB = SX1-1.6
      NN = 0
      IF (KTOTAL-KMIN) 150,150,110
 110  DO 130 J = KMIN, KTOTAL
      INDA = IBEG(J)
      INDB = IEND(J)
      IF (X(INDA)-SXB) 115,125,125
 115  IF (X(INDB)-SXB) 120,125,125
 120  IOMIT(J) = 0
      NN = NN+1
      GO TO 130
 125  IOMIT(J) = 1
 130  CONTINUE
      DO 145 J = KMIN, KTOTAL
      K1 = J
 135  K2 = K1+1
      IF (IOMIT(K2)-IOMIT(K1)) 140,145,145
C
C  THE CDC 6600 IMPLIMENTATION OF THIS ROUTINE ALLOWED
C  ARRAYS IN A COMMON BLOCK TO PASSED TO SUBROUTINES WHICH
C  CHANGED THE VALUES OF THOSE ARRAY ELEMENTS
C  SEVERAL MINOR CHANGES WERE NECESSARY, SUCH AS THE FOLLOWING,
C  TO INSTALL THIS CODE ON THE AMDAHL.
C140  CALL INTER (IOMIT(K1), IOMIT(K2))
C     CALL INTER (NCOUNT(K1), NCOUNT(K2))
C     CALL INTER (IBEG(K1), IBEG(K2))
C     CALL INTER (IEND(K1), IEND(K2))
  140 IOMK1 = IOMIT(K1)
      IOMK2 = IOMIT(K2)
      IOMIT(K1) = IOMK2
      IOMIT(K2) = IOMK1
      NCNTK1 = NCOUNT(K1)
      NCNTK2 = NCOUNT(K2)
      NCOUNT(K1) = NCNTK2
      NCOUNT(K2) = NCNTK1
      IBEGK1 = IBEG(K1)
      IBEGK2 = IBEG(K2)
      IBEG(K1) = IBEGK2
      IBEG(K2) = IBEGK1
      IENDK1 = IEND(K1)
      IENDK2 = IEND(K2)
      IEND(K1) = IENDK2
      IEND(K2) = IENDK1
      K1 = K1-1
      IF (K1-KMIN) 145,135,135
 145  CONTINUE
      KMIN = KMIN+NN
 150  IF (KSTOR-575) 20,20,155
```

```
 155  KSTOR = 576
      KMAX = 575
      RETURN
C****STRING-COLLAPSE ROUTINE FOR PRODUCING CONTINUOS CONTOURS
 160  MAXCHK = KTOTAL+1
      X(600) = .9999E+30
      Y(600) = .9999E+30
      IBEG(MAXCHK) = 600
      IEND(MAXCHK) = 600
      DO 165 I = 1, MAXCHK
 165  IOMIT(I) = I
      I = 0
 170  I = I+1
      IF (KTOTAL-I) 255,175,175
 175  ISTR = IOMIT(I)
      IENDA = IEND(ISTR)
      IBEGA = IBEG(ISTR)
      XIBEGA=X(IBEGA)
      YIBEGA=Y(IBEGA)
      XIENDA=X(IENDA)
      YIENDA=Y(IENDA)
      K = NEQL (XIBEGA , YIBEGA , XIENDA , YIENDA)
      GO TO (170,180), K
 180  XBEG = X(IBEGA)
      YBEG = Y(IBEGA)
      XEND = X(IENDA)
      YEND = Y(IENDA)
      KCHECK = I+1
      MAXCHK = KTOTAL+1
      DO 250 J = KCHECK, MAXCHK
C     J = J
      JSTR = IOMIT(J)
      JENDB = IEND(JSTR)
      JBEGB = IBEG(JSTR)
C****CHECK TO SEE IF TWO STRINGS COINCIDE IN THEIR INITIAL POINTS
      XJBEGB=X(JBEGB)
      YJBEGB=Y(JBEGB)
      K = NEQL(XBEG, YBEG, XJBEGB, YJBEGB)
      GO TO (185,200), K
 185  IF (NCOUNT(ISTR)-NCOUNT(JSTR)) 190,190,195
 190  CALL REVERS (IBEGA, IENDA, ISTR)
      NEXT(IENDA) = NEXT(JBEGB)
      IEND(ISTR) = JENDB
      GO TO 240
 195  CALL REVERS (JBEGB, JENDB, JSTR)
      NEXT(JENDB) = NEXT(IBEGA)
      IBEG(ISTR) = JBEGB
      GO TO 240
C****CHECK TO SEE IF TWO STRINGS COINCIDE IN THEIR TERMINAL POINTS
 200  XJENDB=X(JENDB)
      YJENDB=Y(JENDB)
      K = NEQL(XEND, YEND, XJENDB, YJENDB)
      GO TO (205,220), K
 205  IF (NCOUNT(ISTR)-NCOUNT(JSTR)) 210,210,215
 210  CALL REVERS (IBEGA, IENDA, ISTR)
      NEXT(JENDB) = NEXT(IBEGA)
      IBEG(ISTR) = JBEGB
      GO TO 240
 215  CALL REVERS (JBEGB, JENDB, JSTR)
      NEXT(IENDA) = NEXT(JBEGB)
      IEND(ISTR) = JENDB
      GO TO 240
C****CHECK TO SEE IF TWO STRINGS COINCIDE WITH ORIENTATION PRESENT
 220  XJENDB=X(JENDB)
      YJENDB=Y(JENDB)
      K = NEQL(XBEG, YBEG, XJENDB, YJENDB)
      GO TO (225,230), K
 225  NEXT(JENDB) = NEXT(IBEGA)
      IBEG(ISTR) = JBEGB
      GO TO 240
 230  XJBEGB=X(JBEGB)
      YJBEGB=Y(JBEGB)
      K = NEQL(XEND, YEND, XJBEGB, YJBEGB)
      GO TO (235,250), K
 235  NEXT(IENDA) = NEXT(JBEGB)
      IEND(ISTR) = JENDB
 240  DO 245 JK = J, KTOTAL
 245  IOMIT(JK) = IOMIT(JK+1)
      KTOTAL = KTOTAL-1
      NCOUNT(ISTR) = NCOUNT(ISTR)+NCOUNT(JSTR)
      GO TO 175
 250  CONTINUE
      GO TO 170
 255  CONTINUE
      RETURN
```

```
      END
      SUBROUTINE INTER (NA, NB)
      NSAVE = NA
      NA = NB
      NB = NSAVE
      RETURN
      END
      FUNCTION NEQL (XA, YA, XB, YB)
      IF (ABS(XA-XB).GE.1.E-6) GO TO 15
      IF (ABS(YA-YB).GE.1.E-6) GO TO 15
10    NEQL = 1
      RETURN
15    NEQL = 2
      RETURN
      END
      SUBROUTINE REVERS (IBEGA, IENDA, ISTR)
      DIMENSION X(600), Y(600), NEXT(600)
      DIMENSION IBEG(200), IEND(200), IOMIT(200), NCOUNT(200)
      COMMON /CTCOM1/ X, Y, NEXT, IBEG, IEND, IOMIT, NCOUNT,KTOTAL
      IA = 0
      IB = IBEGA
5     IC = NEXT(IB)
      NEXT(IB) = IA
      IA = IB
      IB = IC
      IF(IC) 10,10,5
10    IS = IBEGA
      IBEGA = IENDA
      IENDA = IS
      IBEG(ISTR) = IBEGA
      IEND(ISTR) = IENDA
      RETURN
      END
```

last specified (italic). Therefore, when you wish to specify another font, if you do not wish to select an italic font, you must specify the upright style.

SWITCHING BETWEEN PRIMARY AND SECONDARY CHARACTER FONTS

Once you have specified your primary and secondary fonts as explained above, you may switch between them using the Shift Out (So) and Shift In (SI) control codes. A Shift Out selects the secondary font and a Shift In selects the primary font.

Raster Graphics

Graphics can be printed on the **LaserJet** printer by sending raster files containing graphics information to the printer. You can print multiple blocks of data (more than one picture) on each page as long the you do not exceed the graphics data limit (59 Kbytes/page). (NOTE: **LaserJet+ printers have expanded memory which allows more graphics data to be printed per page.)**

Each raster line has about 10 bytes of overhead associated with it. This overhead is relatively small for raster lines of 100 bytes or more, but becomes more dominant for very short raster lines. If text and graphics are mixed, the graphics data limit is reduced by approximately four bytes for each character printed.

NOTE

The raster motion is perpendicular to the paper's motion through the printer. **Portrait mode is recommended for printing simple raster graphics applications.** To print raster graphics in the landscape mode, you must first move the cursor to the right (at least the length of the graphics) using the horizontal cursor positioning escape sequences. Otherwise, the printer will attempt to print your graphics in the unprintable region and your data will be lost.

Some programming languages periodically insert Carriage Return and Line Feed commands into the data, causing blank lines in the graphics pattern. These automatic CR and LF commands should be disabled. (For example, the command WIDTH "LPT1:",255 is used with IBM BASIC, version 3.0 to disable the automatic insertion of CR and LF commands. The command WIDTH LPRINT 255 performs the same function with BASIC on the HP 150. Consult your programming manual under the WIDTH command for information on disabling these commands using your programming language.)

To send raster graphics data to the printer, the following four escape sequences are used. The four sequences should be sent to the printer *in the order shown here:*

1) Raster graphics resolution
2) Start raster graphics
3) Transfer raster graphics
4) End raster graphics

The start, transfer, and end raster graphics commands must be sent with *every* block of graphics data you send to the printer. The raster graphics resolution command need only be sent when you wish to change the resolution.

RASTER GRAPHICS RESOLUTION

The **LaserJet** printer has the capability of printing graphics at variable resolution: 300, 150, 100, and 75 dots per inch (DPI). The following escape sequence sets the resolution at which the following graphics data will be printed:

^{E}C*t#R

= Resolution (75, 100, 150, or 300 dpi)

The value field (#) specifies the resolution (75, 100, 150, or 300) in dots per inch. For example, to set the resolution to 300 dots per inch, the ^{E}C*t300R sequence would be sent to the printer.

The default resolution is 75 dots per inch. This command should be sent before the start graphics command. Once the start graphics command is received by the printer, the ^{E}c*t#R command is ignored until the end graphics command is received.

Since the LaserJet graphics memory limit is 59 Kbytes/page, the size of the raster image is limited by the resolution used. **NOTE: 59 Kb is valid for typical pictures. However, attempting to print several independent vertical lines can significantly reduce the total raster image size.** The following table lists the approximate maximum raster image size (in square inches) for each resolution (for LaserJet and LaserJet+):

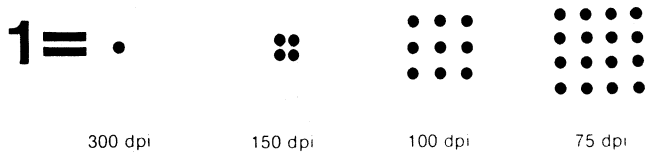
TABLE 2-5. MAXIMUM RASTER IMAGE SIZE

RESOLUTION (dots/inch)	MAXIMUM RASTER IMAGE (square in.)	
	LASERJET	LaserJet+
75	85.9	full page
100	48.3	full page
150	21.5	full page
300	5.4	30 - 32

NOTE

Changing the resolution also changes the size of the printed raster image. At 300 dots/inch, each "1" sent to the printer represents one dot; at lower resolution, however, each "1" represents a number of dots. **NOTE: Data is transferred to the printer as either a "1" (dot) or "0" (no dot).** For example, using 100 dots/inch resolution, each "1" sent to the printer causes 9 dots to be printed. This is because the printer always prints a 300 x 300 (90,000) dot/square inch image. In order to print at 100 dot/inch resolution (10,000 dots/square inch), the printer forms a "dot" from 9 dots (90,000/10,000 = 9). Therefore, your image printed using 100 dots/inch resolution will be 9 times larger than the same image printed using 300 dots/inch. Likewise, the same image will

be 4 times larger (than the 300 dpi image) using 150 dot/inch resolution, and 16 times larger using 75 dot/inch resolution.



START RASTER GRAPHICS

The following escape sequence notifies the printer that raster graphics will follow and also specifies the starting position:

^{E}c*r#A

= 0 or 1

The value field (# = 0 or 1) specifies the starting position. A value field of zero specifies that the graphics should be started at the left-most printable position on the page (not the left margin).

A value field of one specifies that the starting position is the current cursor position and the left graphics margin is set to the current x (horizontal) position. Before you send this command, you may move the cursor (currently active printing position) using the cursor-positioning escape sequences. Then, when you start printing, the upper left corner of the graphics image will be printed where you moved the cursor. NOTE: **When printing graphics in the landscape mode, the "left" graphics margin becomes the margin at the top of the page.**

Once the start raster graphics command is received by the printer, the graphics resolution and left graphics margin are fixed (until the end raster graphics command is received).

TRANSFER RASTER GRAPHICS

The following escape sequence prepares the printer to receive a specific number of bytes of data and transfers the graphics data to the printer:

$^{E}_{C}$*b#W[raster data]

= Number of bytes in this line of data

NOTE: **The brackets in the above escape sequence are used for clarity. Do not include the brackets when sending the escape sequence to the printer.**

The value field (#) indicates the number of bytes of binary data to be sent to the printer. These bytes are interpreted as one line of raster graphics data (one dot row). The data should immediately follow W, the terminating character of the escape sequence. **NOTE: A transfer raster graphics sequence must be sent to the printer for each line (dot row) of graphics data.**

The bits of raster graphics data (1's and 0's) sent to the printer describe single dots to be printed on the page. The most significant bit (bit 7) of the first byte of data corresponds to the first pixel within that line. A one indicates that the dot is to be printed, and a zero indicates that the dot should not be printed.

When this escape sequence is executed, the current active cursor position is at the beginning of the next raster line at the left graphics margin. Each dot of the raster data is expanded according to the specified resolution. Graphics is independent of the text margins and perforation skip mode--it ignores these boundaries. Graphics is only limited by the printable area of the LaserJet printer and the page length.

END RASTER GRAPHICS

The following escape sequence informs the printer that all raster graphics data has been transferred to the printer:

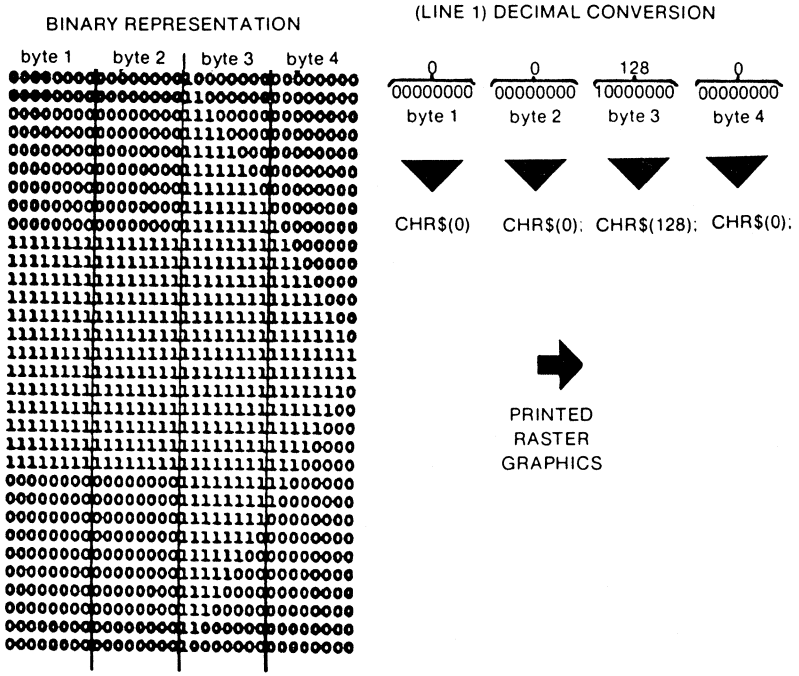
$^{E}_{C}$*rB

After this sequence is sent to the printer, you can then send text to merge with your graphics.

EXAMPLE: RASTER GRAPHICS

The following example program prints raster graphics in the shape of a small arrow. The illustration of 1's and 0's shows how each dot is placed to form the arrow. A "1" causes the printer to print a dot and a zero causes a space. (Actually, each dot in the representation is equivalent to sixteen dots because the resolution is set to 75 dots/inch.)

In BASIC, data must be sent to the printer in decimal numbers (in the form CHR$(decimal number)). Therefore, each group of eight 1's or 0's (each byte) must be converted to a decimal number (see ASCII table in Appendix B). The following illustration shows the binary (1's and 0's) representation of the arrow and illustrates how the first line of binary data is converted to decimal so that BASIC can interpret the data. **NOTE: The example has been tested using MicroSoft BASIC, version D3.0 or A3.0 on the IBM PC. Other versions or releases may or may not support some of the BASIC commands (such as WIDTH "LPT1:",255).**



BASIC PROGRAM TO PRINT ARROW

```
10 REM ***BASIC PROGRAM TO PRINT AN ARROW***
20 REM
30 REM
40 WIDTH "LPT1:",255  :REM **DISABLE AUTO CR-LF**
50 OPEN "LPT1:" AS #1 :REM **OPEN PRINTER AS A FILE**
60 PRINT #1,CHR$(27);"E"; :REM **RESET PRINTER**
70 PRINT #1,CHR$(27);"*t75R"; :REM **SET RESOLUTION**
80 PRINT #1,CHR$(27);"&a40C"; :REM **MOVE CURSOR TO COLUMN 40**
90 PRINT #1,CHR$(27);"*r1A"; :REM **START RASTER GRAPHICS**
100 REM
110 REM **BEGIN LOOP TO READ DATA AND PRINT GRAPHICS**
120 FOR J = 1 TO 32
130 READ A,B,C,D  :REM **EACH RASTER ROW HAS FOUR BYTES**
140 PRINT #1,CHR$(27);"*b4W";CHR$(A);CHR$(B);CHR$(C);CHR$(D);
150 NEXT J
160 REM
170 REM
180 PRINT #1,CHR$(27);"*rB"; :REM **END RASTER GRAPHICS**
190 PRINT #1,CHR$(27);"E"; :REM **RESET PRINTER & EJECT PAGE**
200 CLOSE
210 REM
220 REM **THIS IS THE DATA FOR THE ARROW**
230 DATA  0, 0,128, 0
240 DATA  0, 0,192, 0
250 DATA  0, 0,224, 0
260 DATA  0, 0,240, 0
270 DATA  0, 0,248, 0
280 DATA  0, 0,252, 0
290 DATA  0, 0,254, 0
300 DATA  0, 0,255, 0
310 DATA  0, 0,255,128
320 DATA 255,255,255,192
330 DATA 255,255,255,224
340 DATA 255,255,255,240
350 DATA 255,255,255,248
360 DATA 255,255,255,252
370 DATA 255,255,255,254
380 DATA 255,255,255,255
390 DATA 255,255,255,255
400 DATA 255,255,255,254
410 DATA 255,255,255,252
```



```
420 DATA 255,255,255,248
430 DATA 255,255,255,240
440 DATA 255,255,255,224
450 DATA 255,255,255,192
460 DATA  0, 0,255,128
470 DATA  0, 0,255, 0
480 DATA  0, 0,254, 0
490 DATA  0, 0,252, 0
500 DATA  0, 0,248, 0
510 DATA  0, 0,240, 0
520 DATA  0, 0,224, 0
530 DATA  0, 0,192, 0
540 DATA  0, 0,128, 0
```

Miscellaneous Features

DISPLAY FUNCTIONS

The following two escape sequences are used to enter and exit the display functions mode:

Enter display functions mode ^{E}cY

Exit display functions mode ^{E}cZ

When in the display functions mode, all control codes and escape sequences are printed as blanks (and not executed) with the following exceptions:

- **CR** -- Carriage Return is executed as a Carriage Return and Line Feed
- ^{E}cZ -- This escape sequence is printed as a "**blank**" followed by a **Z**; then the printer exits display functions mode.

The default state is display functions mode off.

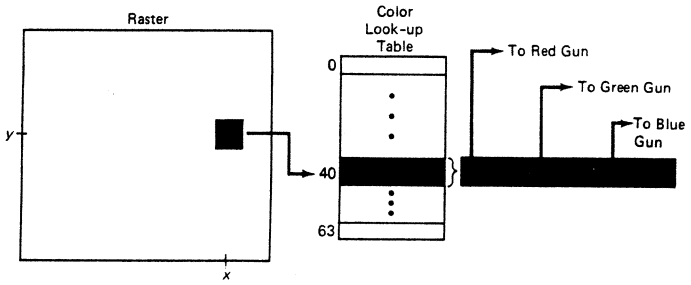


FIGURE 4-3
A color lookup table, providing 12 bits per entry, used with a raster containing 6 bits for each pixel position. A value of 40 stored in raster position *(x, y)* references the location in this lookup table containing the value 356. Each 4-bit segment of this entry controls the intensity level of one of the three electron guns in an RGB monitor, as shown.

for each component pixel in the output primitives to be displayed in that color. The scheme given in Fig. 4–2 allows eight color choices with 3 bits per pixel of storage. Each of the three bit positions is used to control the intensity level (either on or off) of the corresponding electron gun in an RGB monitor. The leftmost bit controls the red gun, the middle bit controls the green gun, and the rightmost bit controls the blue gun. Adding more bits per pixel to the frame buffer increases the number of color choices. In a raster with 6 bits per pixel, 2 bits can be used for each gun. This provides four intensity levels for each color gun (red, green, and blue), and 64 different color codes could be stored.

Another method for storing color codes is diagrammed in Fig. 4–3. In this scheme, the electron gun intensities are controlled by values stored in a **color lookup table** instead of by the values stored in the raster. The raster values are used as indices into the lookup table. For the example shown in this figure, a raster with 6 bits per pixel can reference any one of the 64 positions in the lookup table. Each entry in the table uses 12 bits to specify a color, so that a total of 4096 different colors is now available. Four bits of intensity information is provided for each electron gun. Systems employing this lookup table would permit a user to select any combination of 64 colors from a 4096-color palette. Also, the lookup table entries could be changed at any time, allowing designs, scenes, or graphs displayed on the screen to take on new color combinations.

Color table entries could be set by a user with the command

```
set_color_table (ct, c)
```

Parameter *ct* is used as a color table position number (0 to 64 for the example in Fig. 4–3), and parameter *c* is the code for one of the possible color choices.

Use of a color lookup table can dramatically increase the color options without corresponding increases in raster size. Some high-quality color systems use as many as 24 bits for each position in the color lookup table and 9 bits per pixel in the frame buffer. This allows 512 colors to be used in each display, with over 16 million color choices for the lookup table entries.

Gray Scale

With monitors that have no color capability, the *line_color* command can be used in an application program to set the intensity level, or **gray scale**, for points along displayed lines. Many packages use numeric values within the range of 0 to 1 to set gray scale levels. This allows the package to be adapted to hardware with differing gray scale capabilities.

INTENSITY CODES	STORED INTENSITY VALUES IN THE FRAME BUFFER (Binary Code)		DISPLAYED GRAY SCALE
0.0	0	(00)	Black
0.33	1	(01)	Dark Gray
0.67	2	(10)	Light Gray
1.0	3	(11)	White

FIGURE 4-4
Conversion of intensity values to integer codes for storage in a frame buffer accommodating a gray scale with four levels. Two bits of storage for each pixel position are needed in the frame buffer.

Storing intensity levels in a raster is similar to storing color codes. If only one bit per pixel is provided in the raster, on (white) and off (black) are the only possibilities for the gray scale. Three bits per pixel can accommodate eight different intensity levels. Higher-quality systems might provide 8 or more bits per pixel in the frame buffer for the intensity levels.

Intensity values specified in an application program are converted to appropriate binary codes for storage in the raster. Figure 4–4 illustrates conversion of user specifications to codes for a four-level gray scale. In this example, any intensity input value near 0.33 would store the binary code 01 in the frame buffer and result in a dark gray shading for these pixels. In an alternative scheme, the user specification might be converted directly to the voltage value that produces this gray scale level on the output device in use.

4–3 Area Filling

An advantage of raster systems is their ability to easily store and display areas filled with a color or shading pattern. Fill patterns for such areas are stored as color or intensity values in a frame buffer. Displaying shaded areas on a vector system is considerably more difficult, since area fill requires drawing line segments within the area boundary during each refresh cycle. Various algorithms have been developed for displaying filled areas on raster systems. One method uses the boundary definition to identify which pixels belong to the interior of an area. Other methods start from a position within the area and paint outward from this point.

Scan-Line Algorithm

A scan-line algorithm uses the intersections between area boundaries and scan lines to identify pixels that are inside the area. Figure 4–5 illustrates an area outline and an individual scan line passing through a polygon. Pixel positions along the

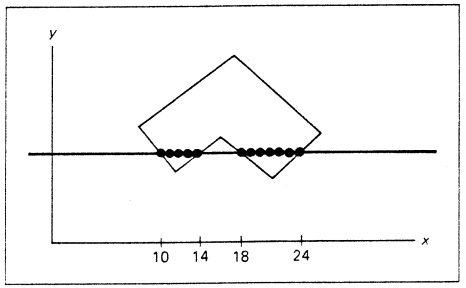


FIGURE 4-5
Interior pixels along a scan line passing through an area to be filled.

scan line that are within the polygon definition are set to the intensity or color values specified in the application program. We first consider how a scan-conversion algorithm can be set up for polygons. The algorithm can then be adapted to other figures, such as circles, by replacing the straight-line equations with the equations defining the figure boundary to be filled.

Taking each scan line in turn, a scan-conversion algorithm locates the intersection points of the scan line with each edge of the area to be filled. Proceeding from left to right, intersections are paired, and the intervening pixels are set to the specified fill intensity or color. In the example of Fig. 4–5, the four intersection points with the polygon boundaries define two stretches of interior pixels.

When a scan line intersects a polygon vertex, it may require special handling. A scan line passing through a vertex intersects two polygon edges at that position, adding two points to the list of intersections for the scan line. In Fig. 4–6, scan line 1 intersects a polygon boundary four times. Two interior stretches are defined: one from the left boundary to the vertex, and a second from the vertex to the right edge of the polygon. But scan line 2 generates five intersections with polygon edges, and resultant pairs do not correspond to the polygon interior.

To fill a polygon correctly, its overall topology must be considered. If the vertices of the polygon are specified in clockwise order, scan line 2 in Fig. 4–6 intersects a vertex whose connecting edges are monotonically decreasing in the y direction. When successive edges of the polygon are monotonically increasing or decreasing, a correct determination of interior points along a scan line is obtained by recording only one intersection point for the vertex. The vertex intersection on scan line 1 in this figure connects two lines with opposite y directions. One line has decreasing y-coordinate values, and the other has increasing y-coordinate values. When such a local minimum (or a local maximum) is encountered by a scan line, two intersection points should be generated to correctly identify interior pixels along the scan line.

Scan-conversion algorithms typically process a polygon from the top of the screen to the bottom and from left to right across each scan line. Calculations performed in such algorithms can be dramatically reduced by making use of various **coherence** properties of the objects being processed. Very often, we can expect properties of pixels along a scan line to be related, so that the properties of one pixel can be determined from those of the preceding pixel. Similarly, we can expect the properties of each scan line to be quite like those of the preceding scan line.

We can take advantage of coherence in calculating scan-line intersections with a polygon by noting that each successive scan line has a y value that is only one

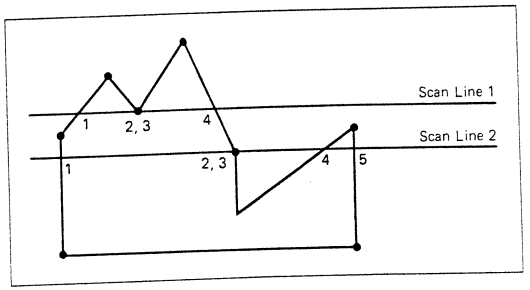


FIGURE 4–6
Intersection points along scan lines that pass through polygon vertices. Scan line 1 generates an even number of intersections that can be paired to correctly identify interior pixels. Scan line 2, however, generates an odd number of intersections.

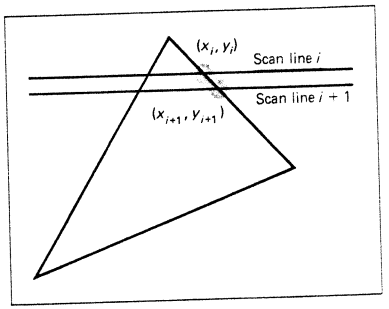


FIGURE 4–7
Two successive scan lines intersecting a polygon boundary.

unit smaller than the previous line. Figure 4–7 shows two successive scan lines crossing a side of a polygon. The slope of this polygon boundary line is

$$m = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

Since the changes in y coordinates is simply

$$y_{i+1} - y_i = -1$$

the x-intersection value x_{i+1} on the lower scan line can be determined from the x-intersection value x_i on the preceding scan line as

$$x_{i+1} = x_i - \frac{1}{m} \tag{4–1}$$

Once the first x-coordinate intersection value x_1 for a polygon side is found for one scan line, we can obtain the x-coordinate values for intersection points on each successive scan line by subtracting the inverse of the slope.

In many cases, we can expect an individual scan line to intersect only some of the total number of sides defining a fill area. To avoid unnecessary checking for intersection points, we can maintain a list of the polygon sides that are crossed by the current scan line. To do this, we create a list of all the polygon sides, sorted on the larger y coordinate of each side. Pointers into this list define the active list of sides for each scan line. Figure 4–8 illustrates the specification of an active list with pointers into the total list of sorted sides. As we move to the next scan line, the pointers are updated to define a new active list.

Procedure *fill_area_solid* incorporates these ideas to fill a defined polygon with a solid color. It accepts parameters specifying the number of vertices in the polygon and the coordinates of the vertices, given in clockwise order. The array of records *sides* stores information about each edge in the polygon. An entry in the array includes the larger y coordinate (*y_top*) of the edge, the length of the edge in the y direction (*delta_y*), and the inverse slope (*x_change_per_scan*). Initially, *x_int* contains the x coordinate of the vertex with the larger y value. When the edge is on the active list, this entry is updated to contain the x coordinate of the edge intersection with the current scan line.

A number of routines are defined within the scope of *fill_area_solid*. Routine *sort_on_bigger_y* creates the edge table and also returns, as parameter *bottomscan*,

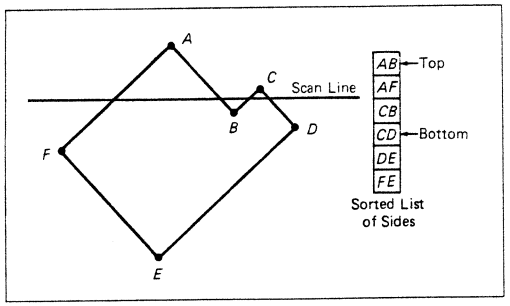


FIGURE 4–8
The active list of sides for a scan line passing through a polygon can be set up as pointers into the sorted list of all polygon sides. Pointers for the scan line shown are labeled Top and Bottom.

the y coordinate of the lowermost vertex. The polygon is filled from the top scan line to *bottomscan*. Routine *update_first_&_last* modifies the pointers *first_s* and *last_s* that define the active list for the current scan line. The number of intersection points for this scan line is determined by *process_x_intersections*, and *draw_lines* pairs the intersections and sets the interior pixels to the fill color. Array *sides* is then updated and made ready for the next scan line.

```
type
  points = array [1..max_points] of integer;

procedure fill_area_solid (count : integer; x, y : points);
  type
    each_entry = record
      y_top : integer;                {larger y coordinate for line}
      x_int : real;                   {x that goes with larger y}
      delta_y : integer;              {difference in y coordinates}
      x_change_per_scan : real        {x change per unit change in y}
      end;  {each_entry}
    list = array [0..max_points] of each_entry;
  var
    sides : list;
    side_count, first_s, last_s, scan, bottomscan, x_int_count, r :
        integer;

  begin {fill_area_solid}
    sort_on_bigger_y (count, x, y, sides, side_count, bottomscan);
    first_s := 1; last_s := 1;   {initialize pointers into sorted list}
    for scan := sides[1].y_top downto bottomscan do begin
        update_first_&_last (sides, count, scan, first_s, last_s);
        process_x_intersections (sides, scan, first_s, last_s,
            x_int_count);
        draw_lines (sides, scan, x_int_count, first_s);
        update_sides_list (sides)
      end  {for scan}
  end;  {fill_area_solid}
```

The first of the second-level routines, *sort_on_bigger_y*, enters edge information in *sides*. Endpoints of each edge are passed to *put_in_sides_list*. In this routine,

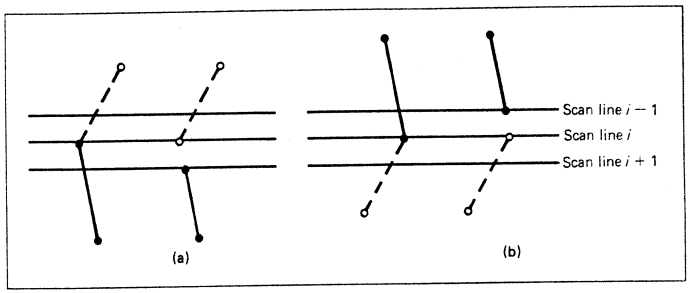


FIGURE 4–9
Adjusting endpoint y values for a polygon edge (solid line). In (a), vertex y coordinates are increasing, so the edge is lowered one unit. In (b), vertex y coordinates are decreasing, so the edge is raised one unit.

a check is made to determine if this edge and the next nonhorizontal edge are monotonically increasing or decreasing. If so, the edge being processed is shortened, to ensure that only one intersection point is generated for the scan line going through the vertex joining the edges. Figure 4–9 illustrates shortening the edge. When the y coordinates of the edges are increasing, the y value of the endpoint is decreased by 1, as in Fig. 4–9 (a). The y value is increased, as in Fig. 4–9 (b), if the y coordinates of the edges are monotonically decreasing. In both cases, the x value of the edge at the adjusted y coordinate is found.

Only nonhorizontal edges are entered into *sides*. An insertion sort based on the larger y coordinate of the edge's endpoints determines placement of the edge in the table. Information stored for the edge are the larger y coordinate and its corresponding x value, the difference between y coordinates of the edge endpoints, and the inverse slope.

```
procedure sort_on_bigger_y (n : integer; x, y : points;
          var sides : list; var side_count, bottomscan : integer);
  var  k, x1, y1, : integer;

  function next_y (k : integer) : integer;
    begin
      {returns y value of the next vertex whose
       y coordinate is not equal to y[k]}
    end;  {next_y}

  procedure put_in_sides_list (var sides : list;
            entry, x1, y1, x2, y2, next_y : integer);
    var
      maxy : integer;
      x2_temp, x_change_temp : real;

    begin
      {make adjustments for problem vertices}
      x_change_temp := (x2 - x1) / (y2 - y1);
      x2_temp := x2;
      if (y2 > y1) and (y2 < next_y) then begin
          y2 := y2 - 1;
          x2_temp := x2_temp - x_change_temp
        end
      else
```

```
      if (y2 < y1) and (y2 > next_y) then begin
         y2 := y2 + 1;
         x2_temp := x2_temp + x_change_temp
       end;
    {insert into sides list}
    if y1 > y2 then maxy := y1 else maxy := y2;
    while (entry > 1) and (maxy > sides[entry - 1].y_top) do begin
       sides[entry] := sides[entry - 1];
       entry := entry - 1
     end;  {while}
    with sides[entry] do begin
       y_top := maxy;
       delta_y := abs(y2 - y1) + 1;
       if y1 > y2 then x_int := x1 else x_int := x2_temp;
       x_change_per_scan := x_change_temp
     end  {with}
  end;  {put_in_sides_list}

begin  {sort_on_bigger_y}
  side_count := 0;
  y1 := y[n]; x1 := x[n];          {initialize}
  bottomscan := y[n];
  for k := 1 to n do begin
     if y1 <> y[k] then begin  {put nonhorizontal edges in table}
        side_count := side_count + 1;
        {pass old point, current point, and}
        {y of next nonhorizontal point}
        put_in_sides_list (sides, side_count, x1, y1, x[k], y[k],
           next_y[k])
      end  {if}
     else  {horizontal}
       {draw x1,y1 to x[k],y1 with fill_color};
     if y[k] < bottomscan then bottomscan := y[k];
     y1 := y[k]; x1 := x[k]    {save for next side}
   end  {for k}
end;  {sort_on_bigger_y}
```

Procedure *update_first_&_last* is called once for each scan line, and it updates the pointers defining the beginning and end of the active list. Figure 4–10 shows a polygon and its list of sorted edges. The pointers *first_s* and *last_s* remain positioned as shown for processing of all lines between the polygon's top scan line and scan line *i*. When processing of scan line *i* is completed, the active list is redefined for the next scan line ($i + 1$). Pointer *last_s* is moved to the table's fifth entry (GA), but *first_s* stays where it is since edge AB intersects scan line $i + 1$. Edge FG is now no longer an active side, but it is embedded within the range of the active-list pointers.

One way to eliminate edge FG from further processing is to shift the entries for edges AB, BC, and EF down one spot in *sides* to maintain a contiguous active list. To avoid shifting edges around in the table, we use the *delta_y* parameter of each edge to indicate the status of that edge in the active list. When this entry has the value 0, the edge is no longer considered active even though it is included within the range of the pointers. This entry is originally set to the difference in *y* values of the two endpoints, and its value is equal to the number of times this edge will intersect

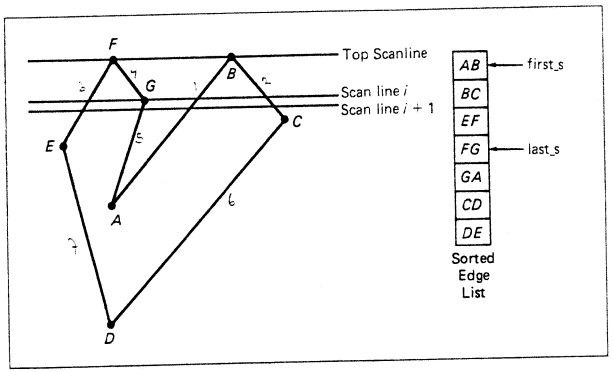


FIGURE 4–10
Processing of pointers for the active list within the sorted edge list for a specified fill area. The sorted edge list is obtained from the input vertices, specified here in the order *A* through *G*. Positions of the pointers first_s and last_s show the active list for all scan lines from the top of the polygon through scan line *i*.

with subsequent scan lines. Each time we process a scan line, *delta_y* is decremented by 1. Active edges, then, are those that have a *delta_y* value greater than 0 and that reside between the *first_s* and *last_s* pointers. The pointer *first_s* is updated only when the *delta_s* value of the entry it points to has become 0.

```
procedure update_first_&_last (sides : list; count, scan : integer;
          var first_s, last_s : integer);
  begin
    while (sides[last_s + 1].y_top >= scan) and
          (last_s < count) do
      last_s := last_s + 1;
    while sides[first_s].delta_y = 0 do
      first_s := first_s + 1
  end;  {update_first_&_last}
```

Taking note of the *delta_y* value for each edge, *process_x_intersections* identifies elements of the active list that intersect the current scan line. These edges are shuffled within the active list to facilitate pairing the intersections that define the polygon interior. The list is reordered on increasing *x_int* values. For example, the active list shown in Fig. 4–10 would assume the new ordering EF, FG, AB, BC. Movement needed within the active list is minimized by the fact that an established ordering will remain constant across a number of scan lines.

The task of identifying pairs of intersections is left to the routine *draw_lines*. This routine takes two successive *x_int* values and sets the intervening pixels on this scan line to the specified fill color.

```
procedure process_x_intersections (var sides : list;
                                   scan, first_s, last_s : integer;
                                   var x_int_count : integer);

  var k : integer;

    {swap is predefined and reverses placement
     of two entries within the table sides}
```

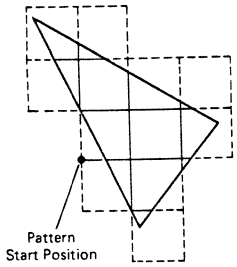
```
procedure sort_on_x (entry, first_s : integer);
  begin
    while (entry > first_s) and
          (sides[entry].x_int < sides[entry-1].x_int) do begin
       swap (sides[entry], sides[entry - 1]);
       entry := entry - 1
    end  {while}
  end;  {sort_on_x}

begin  {process_x_intersections}
  x_int_count := 0;
  for k := first_s to last_s do
    if sides[k].delta_y > 0 then begin
       x_int_count := x_int_count + 1;
       sort_on_x (k, first_s)
    end
end;  {process_x_intersections}

procedure draw_lines (sides : list; scan, x_int_count, index : integer);
  var k, x, x1, x2 : integer;
  begin
    for k := 1 to round(x_int_count / 2) do begin
       while sides[index].delta_y = 0 do index := index + 1;
       x1 := round(sides[index].x_int);
       index := index + 1;
       while sides[index].delta_y = 0 do index := index + 1;
       x2 := round(sides[index].x_int);
       for x := x1 to x2 do
         set_pixel (x, scan, fill_color);
       index := index + 1
    end  {for k}
  end;  {draw_lines}
```

Procedure *update_sides_list* is the last of the second-level routines. Its job is to ready the entries in *sides* for processing of the next scan line. For each edge on the active list, *delta_y* is decremented, and *x_change_per_scan* is subtracted from *x_int*.

```
procedure update_sides_list (var sides : list);
  var k : integer;
  begin
    for k := first_s to last_s do with sides[k] do
      if delta_y > 0 then begin
         {determine next x_int, decrease delta_y}
         delta_y := delta_y - 1;
         x_int := x_int - x_change_per_scan
      end  {if delta_y > 0}
  end;  {update_sides_list}
```

To produce a patterned fill, we modify these scan-line procedures so that a selected pattern is superimposed onto the scan lines. Beginning from a specified start position for a pattern fill, the rectangular patterns would be positioned horizontally across corresponding scan lines and vertically across groups of scan lines. Figure 4–11 illustrates the method for positioning a pattern across a defined area.

FIGURE 4–11
From the start position shown, patterns are laid out so as to cover all scan lines passing through the interior of the specified fill area.



Antialiasing Area Boundaries

The antialiasing concepts we have discussed for lines can be applied to the boundaries of areas to remove the jagged appearance generated on raster systems. We can implement these procedures into a scan-line algorithm to smooth the area outline as the area is generated.

If system capabilities permit the repositioning of pixels, area boundaries can be smoothed by adjusting boundary pixel positions so that they are along the line defining an area boundary. Other methods adjust each pixel intensity at a boundary position according to the percent of pixel area that is inside the boundary. This situation is depicted in Fig. 4–12, where each pixel area is represented as a small rectangle.

In this example, the pixel at position (x, y) has about half its area inside the polygon boundary. Therefore, the intensity at that position would be adjusted to one-half its assigned value. At the next position $(x + 1, y + 1)$ along the boundary, the intensity is adjusted to about one-third the assigned value for that point. Similar adjustments, based on the percent of pixel area coverage, are applied to the other intensity values around the boundary.

Various techniques can be used to estimate the amount of each pixel inside the boundary of an area. One way to estimate this interior area is to subdivide the total area and determine how many subdivisions are inside the boundary. In Fig. 4–13, each pixel area is partitioned into four parts so that the original 5 by 5 grid of pixels is subdivided into a 10 by 10 grid. This turns the original five scan lines covering these pixels into ten scan lines for the subdivisions. A scan-line method can then be used to determine which subdivisions are on or inside the boundary line, as shown in Fig. 4–14. In this example, the two scan lines are processed to determine that three subdivisions are inside the boundary. The pixel intensity, therefore, is adjusted to 75 percent of its assigned value. With this method, the accuracy of intensity setting depends on the number of pixel subdivisions used.

Another method for determining the percent of pixel area within a boundary, developed by Pitteway and Watkinson, is based on Bresenham's line algorithm. This algorithm selects the next pixel along a line by determining which of two pixels is closer to the line, as determined by the sign of a parameter p that measures relative distances of the two pixels from the line. By slightly modifying the form of p, we obtain a quantity that gives the percent of pixel area that is covered by a surface.

We consider the method for a line with slope m in the range from 0 to 1. In Fig. 4–15, a line with equation $y = mx + b$ is shown on a pixel grid. Assuming that the pixel at position (x_i, y_i) has been plotted, the next pixel nearest the line at $x = x_i + 1$ is either the pixel at y_i or the one at $y_i + 1$. We can determine which pixel is nearer with the calculation

$$y - y_{\text{mid}} = [m(x_i + 1) + b] - (y_i + 0.5) \tag{4–2}$$

This gives the distance from the y coordinate on the line to the halfway point between pixels at positions y_i and $y_i + 1$. If this difference calculation is negative, the pixel at y_i is closer to the line. If the difference is positive, the pixel at $y_i + 1$ is closer. We can adjust this calculation so that it produces a positive number in the range from 0 to 1 by adding the quantity $1 - m$:

$$p = [m(x_i + 1) + b] - (y_i + 0.5) + (1 - m) \tag{4–3}$$

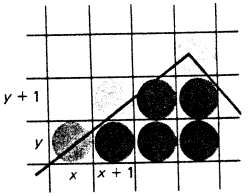


FIGURE 4–12
Adjusting pixel intensities along an area boundary.

FIGURE 4–13
A 5 by 5 pixel section of a raster display subdivided into a 10 by 10 grid.

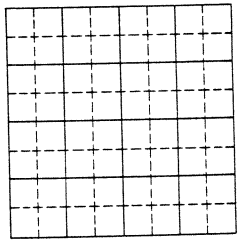
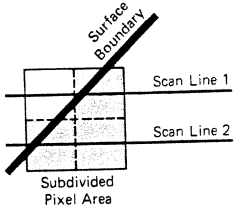


FIGURE 4–14
A subdivided pixel area with three subdivisions inside the boundary line.



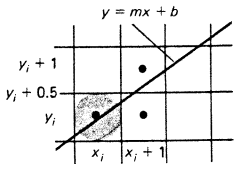


FIGURE 4–15
Boundary line of an area passing through a pixel grid section.

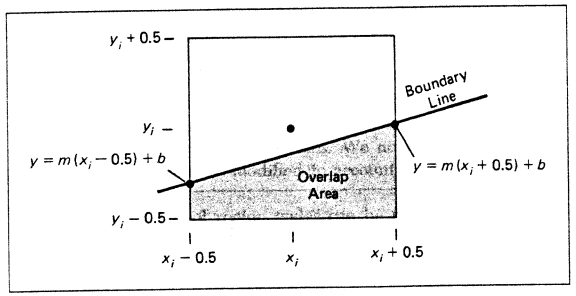


FIGURE 4–16
A pixel represented as a rectangular area with center at (x_i, y_i) has an overlap with a bounded region as indicated by the shaded area.

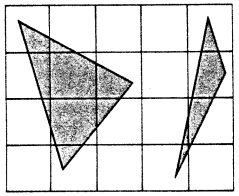
Now the pixel at y_i is nearer if $p < 1 - m$, and the pixel at $y_i + 1$ is nearer if $p > 1 - m$.

Parameter p also measures the amount of a pixel that is overlapped by an area. For the rectangular pixel in Fig. 4–16, the part of the pixel that is inside the boundary line has an area that can be calculated as

$$area = mx_i + b - y_i + 0.5 \tag{4–4}$$

This expression for the overlap area of the pixel at (x_i, y_i) is the same as that for parameter p in Eq. 4–3. Therefore, by evaluating p at each successive pixel position, we can determine the percent of area coverage for the preceding pixel. This value then sets the intensity level for the pixel.

FIGURE 4–17
Areas on a pixel grid that have more than one boundary line passing through a pixel region.



We can incorporate this method into a scan-line algorithm by adjusting the pixel intensities at the points on each scan line that intersect the area boundaries. Also, the parameters in p can be adjusted to accommodate lines with negative slopes and slopes greater than 1. This provides an efficient method for antialiasing boundaries when only one edge of an area passes through a pixel. More than one side of a boundary can cross a single pixel at polygon vertices and when very skinny areas are to be displayed (Fig. 4–17). For these cases, we can apply other methods, such as pixel subdivision.

The various antialiasing methods can be applied to polygon areas or to regions with curved boundaries. In either case, boundary equations are used to determine the position of the boundaries relative to pixel positions. Using a scan-line method, coherence techniques are used to simplify the calculations from one scan line to the next.

Boundary-Fill Algorithm

As an alternative to the scan-line method, an area can be filled by starting at a point inside the figure and painting the interior in a specified color or intensity. Painting proceeds until the figure's boundary is encountered. This method, called the **boundary-fill algorithm,** is useful in interactive sketching and painting packages. Using a graphics tablet or other interactive device, a user sketches a figure outline, picks an interior point, and selects a color or pattern from a color palette. The system then paints the figure interior.

A boundary-fill algorithm accepts as input the coordinates of the interior point (x, y), a fill color, and a boundary color. Starting from (x, y), neighboring points are tested to determine whether they are of the boundary color. If not, they are painted with the fill color. In this way, all points are tested up to the area boundary.

Figure 4–18 shows two methods for proceeding to neighboring points from the interior start point (x, y). In Fig. 4–18 (a), four neighboring points are tested. These are the points that are above, below, to the right, and to the left of the starting point. Areas filled by this method are called **4-connected.** Another approach, shown in Fig. 4–18 (b), is used to fill more complex figures. Here the set of neighboring points also includes the four diagonal pixels. Fill methods using this approach are called **8-connected.** An 8-connected boundary-fill algorithm would correctly fill the interiors of the areas defined in Fig. 4–19, but a 4-connected boundary-fill algorithm produces the partial fill shown.

The following procedure illustrates a recursive method for filling a 4-connected area with an intensity specified in parameter *fill_color* up to a border color specified by *boundary*.

```
{inquire_color is predefined function that
 returns the current color of point (x, y)}

procedure boundary_fill (x, y, fill_color, boundary : integer);
  var present_color : integer;
  begin
    present_color := inquire_color (x, y);
    if (present_color <> boundary) and
      (present_color <> fill_color) then begin
        set_pixel (x, y, fill_color);
        boundary_fill (x + 1, y     , fill_color, boundary);
        boundary_fill (x - 1, y     , fill_color, boundary);
        boundary_fill (x    , y + 1, fill_color, boundary);
        boundary_fill (x    , y - 1, fill_color, boundary);
      end  {if present_color}
  end;  {boundary_fill}
```

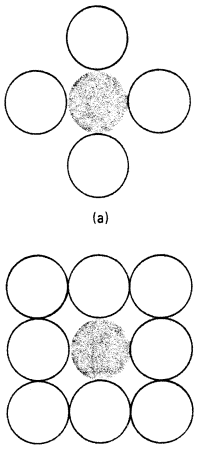


FIGURE 4–18
Boundary-fill methods applied to (a) a 4-connected area and (b) an 8-connected area. Hollow circles represent pixels to be tested from the interior start point (solid).

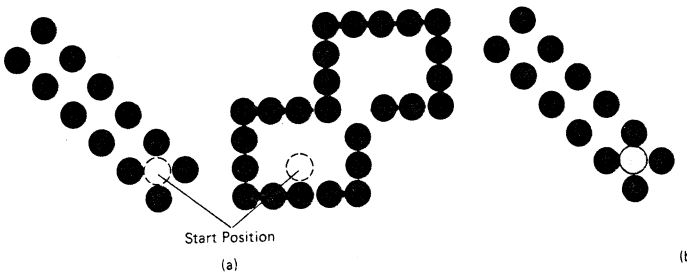


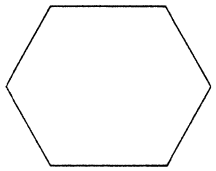
FIGURE 4–19
The defined areas (a) are only partially filled (b) by a 4-connected boundary-fill algorithm.

Since this procedure requires considerable stacking of neighboring points, more efficient methods are generally employed. These methods fill in all points along each horizontal line, one at a time, instead of testing 4-connected or 8-connected neighboring points. The general approach is first to completely fill in the scan line containing the starting point. Then process all lines below the start line, down to the bottom boundaries. Finally, all lines above the start line are processed, up to the top boundaries.

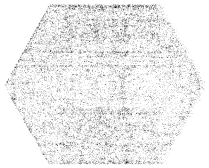
Flood-Fill Algorithm

Another way to fill an area from some start point (x, y) is to specify an interior color value that is to be replaced by the fill color. Instead of looking for a specified boundary color, this **flood-fill algorithm** looks for the interior color that is to be replaced. Using either the 4-connected or 8-connected approach, this procedure stops when no neighboring points have the specified interior color. In a scan-line approach, adjacent pixels with the specified interior color would be set to the fill color until a different color is encountered.

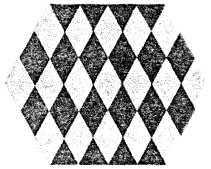
FIGURE 4–20
Interior polygon fill styles.



Hollow
(a)



Solid
(b)



Patterned
(c)

Area-Filling Commands

Several attribute options for fill areas can be provided to a user of a graphics package. These attributes include fill style, fill color, and fill pattern. Fill style refers to the general type of interior, such as hollow or patterned, and fill pattern references the particular pattern code to be used. Parameters specifying these options can be added to the system attribute list and used by the scan-line algorithm to generate fill areas.

A user might select a particular type of interior fill with a command such as

```
set_fill_area_interior_style (fs)
```

Possible types of fill that could be chosen for the fillstyle parameter *fs* are *hollow*, *solid*, and *patterned* (Fig. 4–20). As with line attributes, the fill style parameter would be recorded in the list of system attributes so that areas defined by subsequent commands (such as *fill_area* and *ellipse* would be displayed with this fill style.

Hollow areas are displayed with only a boundary outline. The color for an area outline, or for a solid interior, is chosen with an area color command:

```
set_fill_area_color_index (fc)
```

where fill color *fc* is set to the desired color code.

Patterns can be specified with the command

```
set_fill_area_pattern_index (pi)
```

where the pattern index parameter *pi* is assigned a pattern code. If integer codes are used to select patterns, the following set of statements would fill the area defined in the *fill_area* command with the second pattern type stored in the pattern table:

```
set_fill_area_interior_style (patterned);
set_fill_area_pattern_index (2);
fill_area (n, x, y);
```


A set of standard patterns can be predefined for a user, or a pattern table could be created with user-defined patterns. Commands to create patterns can be made available to a user in the form

```
set_pattern_representation (pi, nx, ny, cp)
```

Parameter *pi* sets the pattern index number, *nx* and *ny* specify the number of points in the x and y directions to be defined in the pattern, and *cp* is a two-dimensional color pattern array of size *nx* by *ny*. The program segment

```
cp[1,1] := 4; cp[2,2] := 4;
cp[1,2] := 0; cp[2,1] := 0;
set_pattern_representation (1, 2, 2, cp);
```

could be used to set the first entry in the pattern table of Fig. 4–21. This pattern, when applied to a defined area, produces alternate red and black diagonal lines.

Positioning patterns on the interior of a fill area is specified with the statement

```
set_pattern_reference_point (xp, yp)
```

The scan-line algorithm uses the coordinates (xp, yp) to fix the lower left corner of the rectangular pattern. From this starting position, the pattern is then mapped onto all scan lines covering the area to be filled. To illustrate the use of the pattern commands, the following program example creates a black-and-white pattern on the interior of the area in Fig. 4–22.

```
type
  points = array [1..max_points] of integer;

procedure fill_triangle;
  var
    pattern : array [1..3,1..3] of integer;
    x, y : points;
```

FIGURE 4–21
A pattern table with two entries, using the color codes of Fig. 4–2.

INDEX (PI)	PATTERN (CP)
1	$\begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix}$
2	$\begin{pmatrix} 2 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 2 \end{pmatrix}$

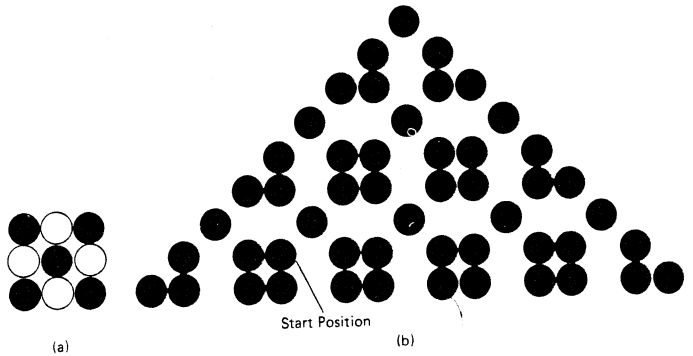


FIGURE 4–22
A defined fill pattern (a) is superimposed on a triangular fill area to produce the patterned display (b).

```
begin
  pattern[1,1] := 1; pattern[1,2] := 0; pattern[1,3] := 1;
  pattern[2,1] := 0; pattern[2,2] := 1; pattern[2,3] := 0;
  pattern[3,1] := 1; pattern[3,2] := 0; pattern[3,3] := 1;
  set_pattern_representation (8, 2, 2, pattern);
  x[1] := 10; y[1] := 10;
  x[2] := 26; y[2] := 10;
  x[3] := 18; y[3] := 18;
  set_fill_area_interior_style (patterned);
  set_fill_area_pattern_index (8);  {use pattern from above}
  set_pattern_reference_point (14, 11);
  fill_area (3, x, y);
end;  {fill_triangle}
```

Another option that can be made available for patterned fill is modification of the pattern size. Graphics packages providing this option include a command to change the dimensions of previously defined pattern arrays. Pattern size can be changed relative to the lower left corner of the array. If a pattern is to be reduced, an appropriate portion of the lower left corner of the array is maintained. For enlargement, the pattern is expanded out from the lower left corner.

4–4 Character Attributes

The appearance of displayed characters is controlled by attributes such as color, style, and orientation. Procedures for displaying character attributes operate on the grid definitions of characters and their placement in the raster. Attributes can be set both for character strings (text) and for additional characters defined as marker symbols.

Text Attributes

Some packages provide a number of choices for text style. The different styles can be made available to the package as predefined sets of grid patterns. Each style can be assigned a text font code, and the character style chosen for display is determined by the current text font setting in the system attribute list. An applications programmer could select a particular code with the text font parameter *tf* in the command

```
set_text_font (tf)
```

Color settings for displayed text are stored in the system attribute list and used by the procedures that load character definitions into the raster array. When a character string is to be displayed, the character grid patterns are entered into the raster with the current color values. Control of text color (or intensity) can be managed from an application program with

```
set_text_color_index (tc)
```

where text color parameter *tc* specifies an allowable color code.

Text size can be adjusted by scaling the height and width of characters. This scaling is applied to the character grid definitions in the horizontal and vertical directions before storage in the frame buffer. Graphics packages can provide separate size adjustment for height and width, or a single attribute parameter can be used. When a single parameter is used, a standard ratio of width to height is maintained. A command for specifying text size is

```
set_character_height (ch)
```

This command determines both the height and width of characters. A value of 1 for the character height parameter *ch* displays characters in the standard size. Values greater than 1 enlarge text, and values smaller than 1 scale down the character sizes.

The orientation for a displayed character string can be set according to the direction of a **character up vector.** Two parameters can be used to specify the slope of this vector, and text is then aligned so that the tops of characters are in the direction of the up vector. A procedure for orienting text would rotate each character string so that the string is stored in the raster in a direction that is perpendicular to the direction of the up vector. Figure 4–23 illustrates the appearance of text oriented by an up vector at 45°. The direction of the up vector is set with the command

```
set_character_up_vector (dx, dy)
```

where the slope of the vector is equal to the ratio *dy*/*dx*.

It is useful in many applications to be able to arrange character strings vertically or horizontally (Fig. 4–24). An attribute parameter for this option could be used to direct a routine to load character patterns into the raster horizontally from the start position or vertically down from the start position. These options could be set by a user with the statement

```
set_text_path (tp)
```

where the text path parameter *tp* is assigned a value of *right* or *down*. Other possible options for text path are *left* and *up*.

Character strings can also be oriented by using a combination of up vector and text path specifications to produce slanted horizontal and vertical text. Figure 4–25 shows the directions of character strings generated by the various text path settings for a 45° up vector. Examples of text generated for *down* and *right* text path values with a 45° up vector are illustrated in Fig. 4–26.

Another handy attribute for character strings is alignment. Figure 4–27 shows various possibilities for text alignment with horizontal and vertical strings. The characters in a string can be aligned on the left, on the right, at the top or bottom, or they can be centered on some position. Once the alignment parameters have been given, procedures for displaying text load the characters into the frame buffer accordingly. A package can provide for alignment with the command

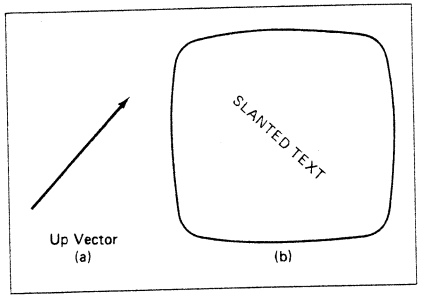


FIGURE 4–23
The direction for the up vector (a) controls the orientation of text (b).

FIGURE 4–24
Text path attributes can be set to produce horizontal or vertical arrangements of character strings.

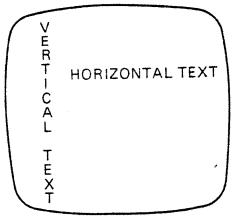
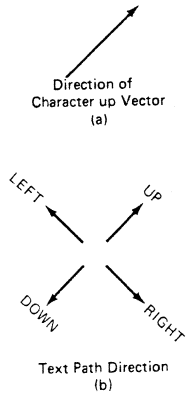


FIGURE 4–25
An up-vector direction (a) can be used to control directions for text path (b).

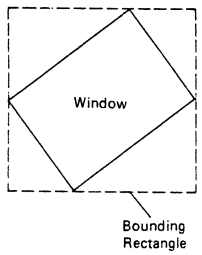


```
            r := q / p;
            if r < u1 then result := false    {reject line or}
            else if r < u2 then u2 := r       {update u2 if appropriate}
          end {if p > 0}
        else                                  {p = 0; line parallel to
                                               boundary}
            if q < 0 then result := false;    {outside of boundary}
      cliptest := result
    end; {cliptest}

  begin {clipper}
    u1 := 0;
    u2 := 1;
    dx := x2 - x1;
    if cliptest (-dx, x1 - xwmin, u1, u2) then
      if cliptest (dx, xwmax - x1, u1, u2) then begin
          dy := y2 - y1;
          if cliptest (-dy, y1 - ywmin, u1, u2) then
            if cliptest (dy, ywmax - y1, u1, u2) then begin
                {if u1 and u2 are within range of 0 to 1,
                 use to calculate new line endpoints}
                if u1 > 0 then begin
                    x1 := x1 + u1 * dx;
                    y1 := y1 + u1 * dy
                  end; {if u1 > 0}
                if u2 < 1 then begin
                    x2 := x1 + u2 * dx;
                    y2 := y1 + u2 * dy
                  end {if u2 < 1}
              end {if cliptest}
          end {if cliptest}
    end; {clipper}
```

The Liang and Barsky line-clipping algorithm reduces the computations that are needed to clip lines. Each update of u_1 and u_2 requires only one division, and window intersections of the line are computed only once, when values of u_1 and u_2 have been finalized. In contrast, the Cohen and Sutherland algorithm repeatedly calculates points of intersection between the line and window boundaries, and each intersection calculation requires both a division and a multiplication.

When rotated windows or arbitrarily shaped polygons are used for windows and viewports, the line-clipping algorithms discussed would require some modification. It is still possible to do preliminary screening of the lines. A rotated window, or any other polygon shape, can be enclosed within a larger rectangle whose sides are parallel to the coordinate axes (Fig. 6–11). Any lines outside the larger bounding rectangle are outside the window also. Inside tests are not so easily done, and intersection points must be calculated using line equations for the window boundaries as well as for the lines to be clipped.

FIGURE 6–11
Rotated window enclosed by a larger bounding rectangle whose boundaries are parallel to coordinate axes.



Area Clipping

Hollow polygons used in line-drawing applications can be clipped by processing each component line through the line-clipping algorithms discussed. A polygon processed in this way is reduced to a series of clipped lines (Fig. 6–12).

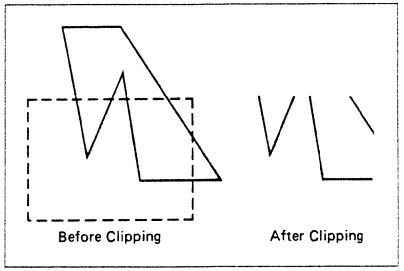


FIGURE 6–12
Polygon clipped by a line-clipping algorithm.

When a polygon boundary defines a fill area, as in Fig. 6–13, a modified version of the line-clipping algorithm is needed. In this case, one or more closed areas must be produced to define the boundaries for area fill.

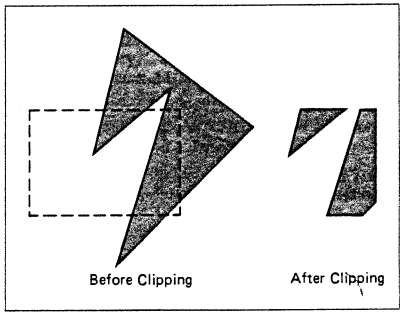


FIGURE 6–13
A shaded area, before and after clipping.

A technique for polygon clipping, developed by Sutherland and Hodgman, performs clipping by comparing a polygon to each window boundary in turn. The output of the algorithm is a set of vertices defining the clipped area that is to be filled with a color or shading pattern. The basic method is illustrated in Fig. 6–14.

Polygon areas are defined by specifying an ordered sequence of vertices. To clip a polygon, we compare each of the vertices in turn against a window boundary. Vertices inside this window edge are saved for clipping against the next boundary;

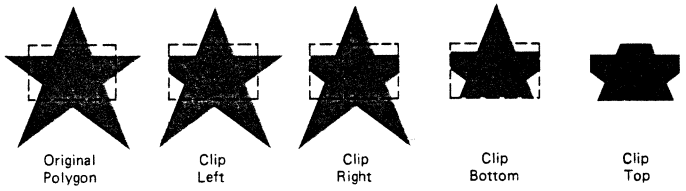
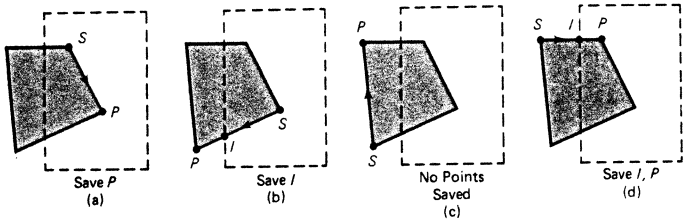


FIGURE 6–14
Clipping a polygon area against successive window boundaries.

FIGURE 6–15
Processing vertices of a polygon relative to a window boundary (dashed lines). From vertex *S*, the next vertex point processed *(P)* may generate one point, no points, or two points to be saved by a clipping algorithm.



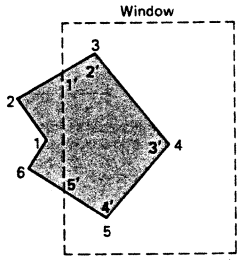
vertices outside the window edge are discarded. If we proceed from a point inside the window edge to an outside point, we save the intersection of the line with the window boundary. Both the intersection and the vertex are saved if we cross from the outside of a window edge to the inside. The four possible situations that can occur as we process a point *(P)* and the previous point *(S)* against a window boundary are illustrated in Fig. 6–15. A point inside the window boundary is saved (case a), while an outside point is not (case c). If a point *P* and the previous point *S* are on opposite sides of a boundary, the intersection *I* is calculated and saved (cases b and d). In case d, the point *P* is inside and the previous point *S* is outside so both the intersection *I* and the point *P* are saved. Once all vertices have been processed for the left window boundary, the set of saved points is clipped against the next window boundary.

We illustrate this method by processing the area in Fig. 6–16 against the left window boundary. Vertices 1 and 2 are found to be on the outside of the boundary. Passing to vertex 3, which is inside, we calculate the intersection and save both the intersection point and vertex 3. Vertices 4 and 5 are determined to be inside, and they also are saved. The sixth and final vertex is outside, so we find and save the intersection point. Using the five saved points, we repeat the process for the next window boundary.

Implementing the algorithm as described necessitates the use of extra storage space for the saved points. This can be avoided if we take each point that would be saved and immediately pass it to the clipping routine, along with instructions to clip it against the next boundary. We save a point (either an original vertex or a calculated intersection) only after it has been processed against all boundaries. It is as if we have a pipeline of clipping routines, with each stage in the pipeline clipping against a different window boundary. A point that is inside or on the window boundary at one stage is passed along to the next stage. A point that is outside at some stage simply does not continue in the pipeline.

The following procedure demonstrates this approach. An array, *s*, records the most recent point that was clipped for each window edge. The main routine passes each vertex *p* to the *clip_this* routine for clipping against the first window edge. If the line defined by endpoints *p* and *s*[*edge*] crosses this window edge, the intersection is found and is passed to the next clipping stage. If *p* is inside the window, it is passed to the next clipping stage. Any point that survives clipping against all window edges is then entered into the output arrays *x_out* and *y_out*. The array *first_point* stores for each window edge the first point that is clipped against that edge. After all polygon vertices have been processed, a closing routine clips lines defined by the first and last points clipped against each edge.

FIGURE 6–16
Clipping a polygon against the left edge of a window, starting with vertex 1. Primed numbers are used to label the points saved by the clipping algorithm.




```
type
  points = array [1..max_points] of real;

procedure polygon_clip (n : integer; x, y : points; var m : integer;
                        var x_out, y_out : points);
  const
    boundary_count = 4;
  type
    vertex = array [1..2] of real;
    boundary_range = 1..boundary_count;
  var
    k : integer;
    p : vertex;
    s, first_point : array [1..boundary_count] of vertex;
    new_edge : array [1..boundary_count] of boolean;

  function inside (p : vertex; edge : boundary_range) : boolean;
    begin
      {returns true if vertex p is inside of window edge}
    end;  {inside}

  function cross (p, s : vertex; edge : integer) : boolean;
    begin
      {returns true if polygon side ps intersects window edge}
    end;  {cross}

  procedure output_vertex (p : vertex);
    begin
      m := m + 1;
      x_out[m] := p[1]; y_out[m] := p[2]
    end;  {output_vertex}

  procedure find_intersection (p, s : vertex;
                               edge : boundary_range; var i : vertex);
    begin
      {returns in parameter i the intersection of ps with window edge}
    end;  {intersection}

  procedure clip_this (p : vertex; edge : boundary_range);
    var i : vertex;
    begin  {clip_this}
      {save the first point clipped against a window edge}
      if new_edge[edge] then begin
          first_point[edge] := p;
          new_edge[edge] := false
        end  {new_edge}
      else
        {if ps crosses window edge, find intersection,
         clip intersection against next window edge}
        if cross (p, s[edge], edge) then begin
            find_intersection (p, s[edge], edge, i);
            if edge < boundary_count then clip_this (i, edge + 1)
            else output_vertex (i)
            end;  {if p & s cross edge}
      {update saved vertex}
```

```
    s[edge] := p;
    {if p is inside this window edge,
     clip it against next window edge}
    if inside (p, edge) then
      if edge < boundary_count then clip_this (p, edge + 1)
      else output_vertex (p)
  end; {clip_this}

procedure clip_closer;
    {closing routine. For each window edge, clips the
     line connecting the last saved vertex and the first_point
     processed against the edge}
    var
      i : vertex;
      edge : integer;
    begin
      for edge := 1 to boundary_count do
          if cross (s[edge], first_point[edge], edge) then begin
               find_intersection (s[edge], first_point[edge], edge, i);
               if edge < boundary_count then clip_this (i, edge + 1)
               else output_vertex (i)
               end  {if s and first_point cross edge}
               end;  {clip_closer}

begin  {polygon_clip}
  m := 0;                              {number of output vertices}
  for k := 1 to boundary_count do
    new_edge[k] := true;
  for k := 1 to n do begin             {puts each vertex into pipeline}
      p[1] := x[k]; p[2] := y[k];
      clip_this (p, 1)                 {clip against first window edge}
    end;  {for k}
  clip_closer                          {close polygon}
end;  {polygon_clip}
```

When a concave polygon is clipped against a rectangular window, the final clipped area may actually represent two or more distinct polygons. Since this area-clipping algorithm produces only one list of vertices, these separate areas will be joined with connecting lines. An example of this effect is shown in Fig. 6–17. Special considerations can be given to such cases to remove the extra lines, or more general clipping algorithms can be employed.

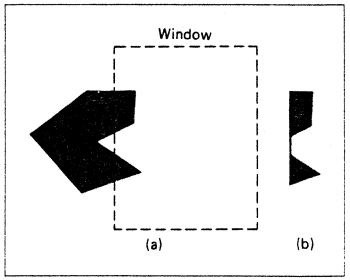


FIGURE 6–17
Clipping the concave polygon in (a) against the window generates the two connected areas in (b).

Although we have limited our discussion to rectangular windows aligned with the x and y axes, we could implement this algorithm to clip against a window of any polygon shape. We would need to store information about each of the window boundaries, and we would need to modify the routines *inside* and *find_intersection* to handle arbitrary boundaries.

Another approach to polygon-area clipping is to employ parametric-equation methods. Arbitrarily shaped windows would then be processed by using parametric line equations to describe both the window boundaries and the boundaries of the areas to be clipped.

Clipping areas other than polygons requires a little more work, since the area boundaries are not defined with straight-line equations. In Fig.6–18, for example, circle equations are needed to find the two intersection points on the window boundary.

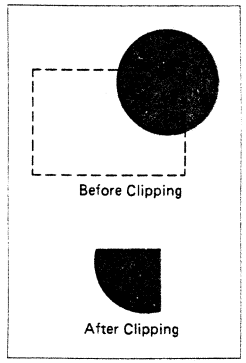


FIGURE 6–18
Clipping a circular area.

Text Clipping

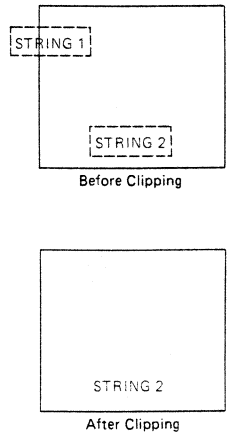
There are several techniques that can be used to provide text clipping in a graphics package. The particular implementation chosen will depend on the methods used to generate characters and the amount of sophistication needed by a user of the package in the processing of text.

The simplest method for processing character strings relative to a window boundary is to use the "all-or-none text-clipping" strategy shown in Fig. 6–19. If all of the string is inside a window, we keep it. Otherwise, the string is discarded. This procedure can be implemented by considering a bounding rectangle around the text pattern. The boundary positions of the rectangle are then compared to the window boundaries, and the string is rejected if there is any overlap. This method produces the fastest text clipping.

An alternative to rejecting an entire character string that overlaps a window boundary is to use the "all-or-none character-clipping" strategy. Here we discard only those characters that are not completely inside the window (Fig. 6–20). In this case, the boundary limits of individual characters are compared to the window. Any character that either overlaps or is outside a window boundary is clipped.

A final method for handling text clipping is to clip individual characters. We now treat characters in much the same way that we treated lines. If an individual character overlaps a window boundary, we clip off the parts outside the window (Fig. 6–21). Characters formed with line segments can be processed in this way using a line-clipping algorithm. Processing characters formed with bit maps requires clipping individual pixels by comparing the relative position of the grid patterns to the window boundaries.

FIGURE 6–19
Text clipping using bounding rectangles. Any rectangles overlapping the boundary are entirely discarded.



Blanking

Instead of saving information inside a defined region, a window area can be used for **blanking** (erasing) anything within its boundaries. What is saved is outside.

Blanking all output primitives within a defined area is a convenient means for overlapping different pictures. This technique is often used for designing page layouts in advertising or publishing applications or for adding labels or design patterns to a picture. The technique can also be used for combining graphs, maps, or schematics. Figure 6–22 illustrates some applications of blanking.

When two displays are to be overlaid using blanking methods, one display can be thought of as the foreground and the other as the background. A blanking window, encompassing the foreground display area, is superimposed on the back-