Chapter 15

Graphics Examples

Contents

Overview

SAS/IML software provides you with a powerful set of graphics commands, called graphics *primitives*, from which to create customized displays. Basic drawing statements include the GDRAW subroutine, which draws a line, the GPOINT subroutine, which plots points, and the GPOLY subroutine, which draws a polygon. With each drawing statement, you can associate a set of attributes such as color or line style.

In this chapter you learn about

- plotting simple two-dimensional plots
- naming and saving a graph
- changing attributes such as color and line style
- specifying the location and scale of your graph
- adding axes and text

SAS/IML graphics commands depend on the libraries and device drivers distributed with SAS/GRAPH software, and they do not work unless you have SAS/GRAPH software.

An Introductory Graph

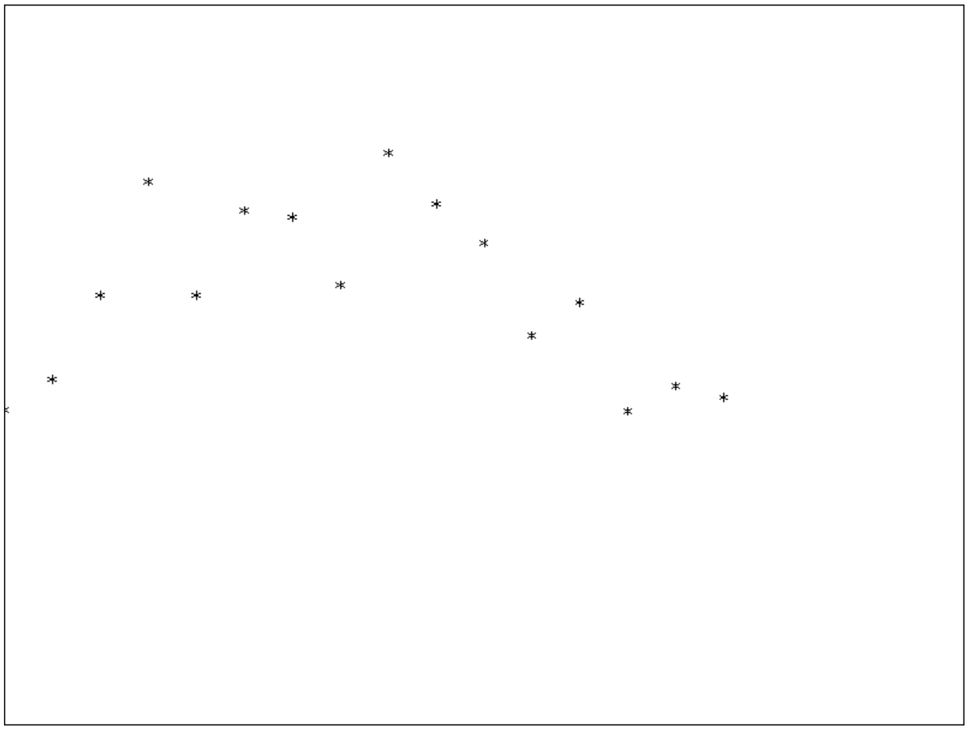
Suppose that you have data for ACME Corporation's stock price and you want a simple PRICE × DAY graph to see the overall trend of the stock's price. The data are as follows.

Day	Price
0	43.75
5	48.00
10	59.75
15	75.5
20	59.75
25	71.50
30	70.575
35	61.125
40	79.50
45	72.375
50	67.00
55	54.125
60	58.750
65	43.625
70	47.125
75	45.50

To graph a scatter plot of these points, enter the following statements. These statements generate Figure 15.1.

```
proc iml;                           /* invoke IML        */
   call gstart;                     /* start graphics    */
   xbox={0 100 100 0};
   ybox={0 0 100 100};
   day=do(0,75,5);                  /* initialize day    */
   price={43.75,48,59.75,75.5,      /* initialize price  */
       59.75,71.5,70.575,
       61.125,79.5,72.375,67,
       54.125,58.75,43.625,
       47.125,45.50};
   call gopen;                      /* start new graph   */
   call gpoly(xbox,ybox);      /* draw a box around plot */
   call gpoint(day,price);          /* plot the points   */
   call gshow;                      /* display the graph */
```

Figure 15.1 Scatter plot



Note that the GSTART statement initializes the graphics session. It usually needs to be called only once. Next, you enter the data matrices. Then you open a graphics segment (that is, begin a new graph) with the GOPEN command. The GPOINT command draws the scatter plot of points of DAY versus PRICE. The GSHOW command displays the graph.

Notice also that, for this example, the x coordinate of the data is DAY and that $0 \leq \text{DAY} \leq 100$. The y coordinate is PRICE, which ranges from $0 \leq \text{PRICE} \leq 100$. For this example, the ranges are this way because the IML default ranges are from 0 to 100 on both the x and y axes. Later on you learn how to change the default ranges for the axes with the GWINDOW statement so that you can handle data with any range of values.

Of course, this graph is quite simple. By the end of this chapter, you will know how to add axes and titles, scale axes, and connect the points with lines.

Details

Graphics Segments

A graph is saved in what is called a graphics segment. A *graphics segment* is simply a collection of primitives and their associated attributes that creates a graph.

Each time you create a new segment, it is named and stored in a SAS graphics catalog called WORK.GSEG. If you want to store your graphics segments in a permanent SAS catalog, do this with options to the GSTART call. You can name the segments yourself in the GOPEN statement, or you can let the IML procedure automatically generate a segment name. In this way, graphics segments that are used several times can be included in subsequent graphs by using the GINCLUDE command with the segment name. You can also manage and replay a segment by using the GREPLAY procedure as well as replay it in another IML session by using the GSHOW command.

To name a segment, include the name as an argument to the GOPEN statement. For example, to begin a new segment and name it STOCK1, use the following statement:

```
   call gopen("stock1");
```

For more information about SAS catalogs and graphics, refer to the chapter on graphics in *SAS/GRAPH Software: Reference*.

Segment Attributes

A set of attributes is initialized for each graphics segment. These attributes are color, line style, line thickness, fill pattern, font, character height, and aspect ratio. You can change any of these attributes for a graphics segment by using the GSET command. Some IML graphics commands take optional attribute arguments. The values of these arguments affect only the graphics output associated with the call.

The IML graphics subsystem uses the same conventions that SAS/GRAPH software uses in setting the default attributes. It also uses the options set in the GOPTIONS statement when applicable. The SAS/IML default values for the GSET command are given by their corresponding GOPTIONS default values. To change the default, you need to issue a GOPTIONS statement. The GOPTIONS statement can also be used to set graphics options not available through the GSET command (for example, the ROTATE option).

For more information about GOPTIONS, refer to the chapter on the GOPTIONS statement in *SAS/GRAPH Software: Reference*.

Coordinate Systems

Each IML graph is associated with two independent cartesian coordinate systems, a *world coordinate System* and a *normalized coordinate system*.

Understanding World Coordinates

The *world coordinate system* is the coordinate system defined by your data. Because these coordinates help define objects in the data's two-dimensional world, these are referred to as *world coordinates*. For example, suppose that you have a data set that contains heights and weights and that you are interested in plotting height versus weight. Your data induces a world coordinate system in which each point (x, y) represents a pair of data values (*height,weight*). The world could be defined by the observed ranges of heights and weights, or it could be enlarged to include a range of all reasonable values for heights and weights.

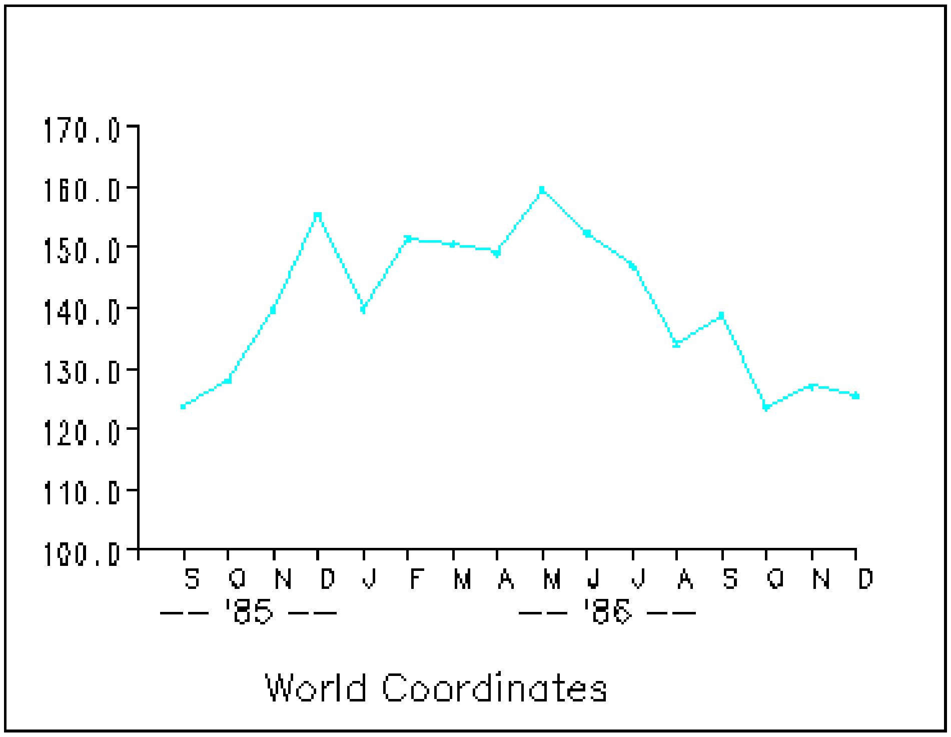
Now consider a more realistic example of the stock price data for ACME Corporation. Suppose that the stock price data were actually the year-end prices of ACME stock for the years 1971 through 1986, as follows:

YEAR	PRICE
71	123.75
72	128.00
73	139.75
74	155.50
75	139.75
76	151.50
77	150.375
78	149.125
79	159.50
80	152.375
81	147.00
82	134.125
83	138.75
84	123.625
85	127.125
86	125.500

The actual range of YEAR is from 71 to 86, and the range of PRICE is from $123.625 to $159.50. These are the ranges in world coordinate space for the stock data. Of course, you could say that the range for PRICE could start at $0 and range upwards to, for example, $200. Or, if you were interested only in prices during the 80's, you could say the range for PRICE is from $123.625 to $152.375. As you see, it all depends on how you want to define your world.

Figure 15.2 shows a graph of the stock data with the world defined as the actual data given. The corners of the rectangle give the actual boundaries for this data.

Figure 15.2 World Coordinates



Understanding Normalized Coordinates

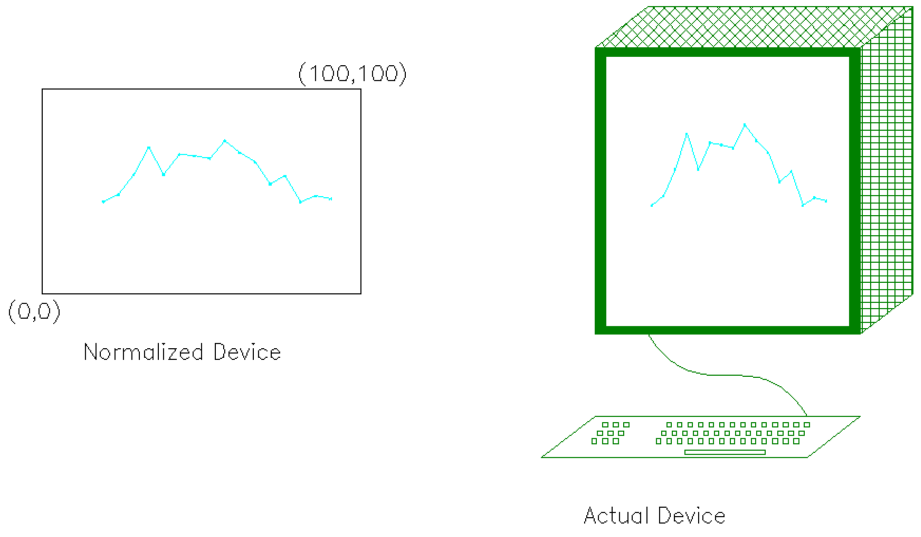
The *normalized coordinate system* is defined relative to your display device, usually a monitor or plotter. It is always defined with points varying between (0,0) and (100,100), where (0,0) refers to the lower-left corner and (100,100) refers to the upper-right corner.

In summary,

- the world coordinate system is defined relative to your data
- the normalized coordinate system is defined relative to the display device

Figure 15.3 shows the ACME stock data in terms of normalized coordinates. There is a natural mathematical relationship between each point in world and normalized coordinates. The normalized device coordinate system is mapped to the device display area so that (0,0), the lower-left corner, corresponds to (71, 123.625) in world coordinates, and (100,100), the upper-right corner, corresponds to (86,159.5) in world coordinates.

Figure 15.3 Normalized Coordinates



Windows and Viewports

A *window* defines a rectangular area in world coordinates. You define a window with a GWINDOW statement. You can define the window to be larger than, the same size as, or smaller than the actual range of data values, depending on whether you want to show all of the data or only part of the data.

A *viewport* defines in normalized coordinates a rectangular area on the display device where the image of the data appears. You define a viewport with the GPORT command. You can have your graph take up the entire display device or show it in only a portion, say the upper-right part.

Mapping Windows to Viewports

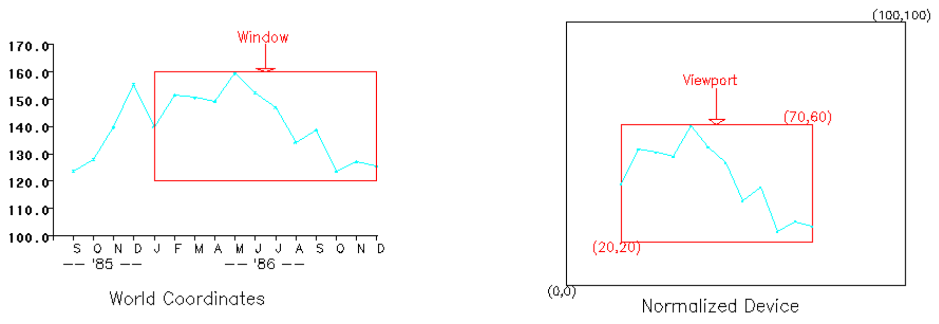
A *window* and a *viewport* are related by the linear transformation that maps the window onto the viewport. A line segment in the window is mapped to a line segment in the viewport such that the relative positions are preserved.

You do not have to display all of your data in a graph. In Figure 15.4, the graph on the left displays all of the ACME stock data, and the graph on the right displays only a part of the data. Suppose that you wanted to graph only the last 10 years of the stock data—say, from 1977 to 1986. You would want to define a window where the YEAR axis ranges from 77 to 86, while the PRICE axis could range from 120 to 160. Figure 15.4 shows stock prices in a window defined for data from 1977 to 1986 along the horizontal direction and from 120 to 160 along the vertical direction. The window is mapped to a viewport defined by the points (20,20) and (70,60). The appropriate GPORT and GWINDOW specifications are as follows:

```
call gwindow({77 120, 86 160});
call gport({20 20, 70 60});
```

The window, in effect, defines the portion of the graph that is to be displayed in world coordinates, and the viewport specifies the area on the device on which the image is to appear.

Figure 15.4 Window to Viewport Mapping



Understanding Windows

Because the default world coordinate system ranges from (0,0) to (100,100), you usually need to define a *window* in order to set the world coordinates that correspond to your data. A window specifies which part of the data in world coordinate space is to be shown. Sometimes you want all of the data shown; other times, you want to show only part of the data.

A window is defined by an array of four numbers, which define a rectangular area. You define this area by specifying the *world coordinates* of the lower-left and upper-right corners in the GWINDOW statement, which has the following general form:

CALL GWINDOW *(minimum-x minimum-y maximum-x maximum-y)* **;**

The argument can be either a matrix or a literal. The order of the elements is important. The array of coordinates can be a 2×2, 1×4, or 4×1 *matrix*. These coordinates can be specified as matrix literals or as the name of a numeric matrix that contains the coordinates. If you do not define a window, the default is to assume both x and y range between 0 and 100.

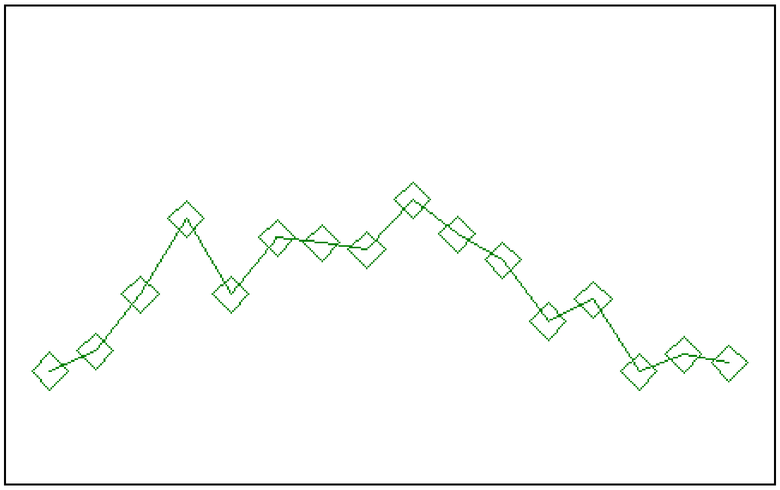
In summary, a window

- defines the portion of the graph that appears in the viewport
- is a rectangular area
- is defined by an array of four numbers
- is defined in world coordinates
- scales the data relative to world coordinates

In the previous example, the variable YEAR ranges from 1971 to 1986, while PRICE ranges from 123.625 to 159.50. Because the data do not fit nicely into the default, you want to define a window that reflects the ranges of the variables YEAR and PRICE. To draw the graph of these data to scale, you can let the YEAR axis range from 70 to 87 and the PRICE axis range from 100 to 200. Use the following statements to draw the graph, shown in Figure 15.5.

```
call gstart;
xbox={0 100 100 0};
ybox={0 0 100 100};
call gopen("stocks1");        /* begin new graph STOCKS1 */
call gset("height", 2.0);
year=do(71,86,1);                     /* initialize YEAR */
price={123.75 128.00 139.75          /* initialize PRICE */
       155.50 139.750 151.500
       150.375 149.125 159.500
       152.375 147.000 134.125
       138.750 123.625 127.125
       125.50};
call gwindow({70 100 87 200});                /* define window */
call gpoint(year,price,"diamond","green"); /* graph the points */
call gdraw(year,price,1,"green");           /* connect points */
call gshow;                                    /* show the graph */
```

Figure 15.5 Stock Data



In the following example, you perform several steps that you did not do with the previous graph:

- You associate the name STOCKS1 with this graphics segment in the GOPEN command.
- You define a window that reflects the actual ranges of the data with a GWINDOW command.
- You associate a plotting symbol, the diamond, and the color green with the GPOINT command.
- You connect the points with line segments with the GDRAW command. The GDRAW command requests that the line segments be drawn in style 1 and be green.

Understanding Viewports

A *viewport* specifies a rectangular area on the display device where the graph appears. You define this area by specifying the *normalized* coordinates, the lower-left corner and the upper-right corner, in the GPORT statement, which has the following general form:

CALL GPORT *(minimum-x minimum-y maximum-x maximum-y)* **;**

The argument can be either a matrix or a literal. Note that both x and y must range between 0 and 100. As with the GWINDOW specification, you can give the coordinates either as a matrix literal enclosed in braces or as the name of a numeric matrix that contains the coordinates. The array can be a 2×2, 1×4, or 4×1 matrix. If you do not define a viewport, the default is to span the entire display device.

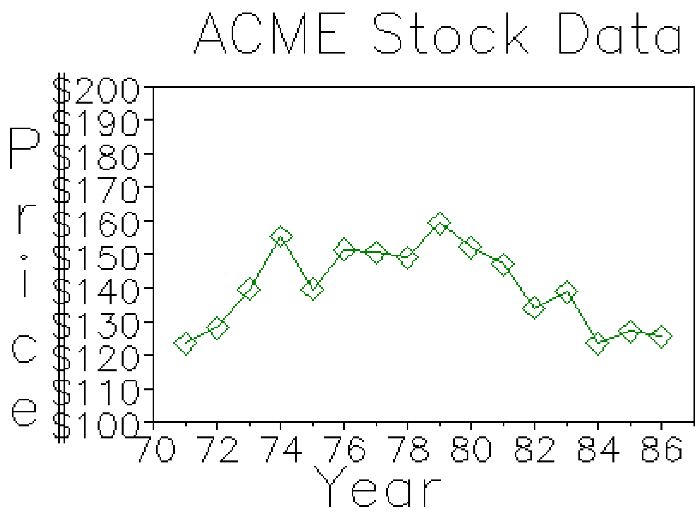
In summary, a viewport

- specifies where the image appears on the display
- is a rectangular area
- is specified by an array of four numbers
- is defined in normalized coordinates
- scales the data relative to the shape of the viewport

To display the stock price data in a smaller area on the display device, you must define a viewport. While you are at it, add some text to the graph. You can use the graph that you created and named STOCKS1 in this new graph. The following statements create the graph shown in Figure 15.6.

```
    /* module centers text strings */
start gscenter(x,y,str);
   call gstrlen(len,str);                 /* find string length */
   call gscript(x-len/2,y,str);                   /* print text */
finish gscenter;

call gopen("stocks2");                    /* open a new segment */
call gset("font","swiss");                /* set character font */
call gpoly(xbox,ybox);                    /* draw a border      */
call gwindow({70 100,87 200});            /* define a window    */
call gport({15 15,85 85});                /* define a viewport  */
call ginclude("stocks1");            /* include segment STOCKS1 */
call gxaxis({70 100},17,18, ,             /* draw x-axis        */
             ,"2.",1.5);
call gyaxis({70 100},100,11, ,                   /* draw y-axis */
             ,"dollar5.",1.5);
call gset("height",2.0);                /* set character height */
call gtext(77,89,"Year");              /* print horizontal text */
call gvtext(68,200,"Price");            /* print vertical text  */
call gscenter(79,210,"ACME Stock Data");      /* print title    */
call gshow;
```

Figure 15.6 Stock Data with Axes and Labels



The following list describes the statements that generated this graph:

- GOPEN begins a new graph and names it STOCKS2.
- GPOLY draws a box around the display area.
- GWINDOW defines the world coordinate space to be larger than the actual range of stock data values.
- GPORT defines a viewport. It causes the graph to appear in the center of the display, with a border around it for text. The lower-left corner has coordinates (15,15) and the upper-right corner has coordinates (85,85).
- GINCLUDE includes the graphics segment STOCKS1. This saves you from having to plot points you have already created.
- GXAXIS draws the x axis. It begins at the point (70,100) and is 17 units (years) long, divided with 18 tick marks. The axis tick marks are printed with the numeric 2.0 format, and they have a height of 1.5 units.
- GYAXIS draws the y axis. It also begins at (70,100) but is 100 units (dollars) long, divided with 11 tick marks. The axis tick marks are printed with the DOLLAR5.0 format and have a height of 1.5 units.
- GSET sets the text font to be Swiss and the height of the letters to be 2.0 units. The height of the characters has been increased because the viewport definition scales character sizes relative to the viewport.
- GTEXT prints horizontal text. It prints the text string `Year` beginning at the world coordinate point (77,89).

- GVTEXT prints vertical text. It prints the text string **`Price`** beginning at the world coordinate point (68,200).
- GSCENTER runs the module to print centered text strings.
- GSHOW displays the graph.

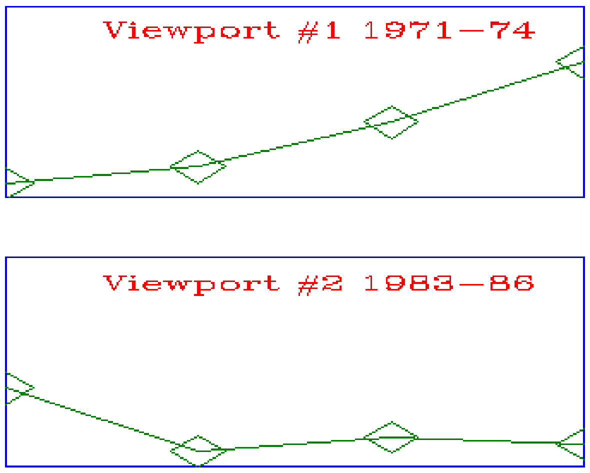
Changing Windows and Viewports

Windows and viewports can be changed for the graphics segment any time that the segment is active. Using the stock price example, you can first define a window for the data during the years 1971 to 1974 and map this to the viewport defined on the upper half of the normalized device; then you can redefine the window to enclose the data for 1983 to 1986 and map this to an area in the lower half of the normalized device. Notice how the shape of the viewport affects the shape of the curve. Changing the viewport can affect the height of any printed characters as well. In this case, you can modify the HEIGHT parameter.

The following statements generate the graph in Figure 15.7:

```
     /* figure 12.7  */
reset clip;                              /* clip outside viewport */
call gopen;                                 /* open a new segment */
call gset("color","blue");
call gset("height",2.0);
call gwindow({71 120,74 175});                 /* define a window */
call gport({20 55,80 90});                   /* define a viewport */
call gpoly({71 74 74 71},{120 120 170 170}); /* draw a border */
call gscript(71.5,162,"Viewport #1 1971-74",,   /* print text */
               ,3.0,"complex","red");
call gpoint(year,price,"diamond","green");      /* draw points */
call gdraw(year,price,1,"green");            /* connect points */
call gblkvpd;
call gwindow({83 120,86 170});            /* define new window */
call gport({20 10,80 45});              /* define new viewport */
call gpoly({83 86 86 83},{120 120 170 170});   /* draw border */
call gpoint(year,price,"diamond","green");      /* draw points */
call gdraw(year,price,1,"green");            /* connect points */
call gscript(83.5,162,"Viewport #2 1983-86",,   /* print text */
               ,3.0,"complex","red");
call gshow;
```

Figure 15.7 Multiple Viewports



The RESET CLIP command is necessary because you are graphing only a part of the data in the window. You want to clip the data that falls outside of the window. See the section "Clipping Your Graphs" on page 425 for more about clipping. In this graph, you

- open a new segment (GOPEN)
- define the first window for the first four years' data (GWINDOW)
- define a viewport in the upper part of the display device (GPORT)
- draw a box around the viewport (GPOLY)
- add text (GSCRIPT)
- graph the points and connect them (GPOINT and GDRAW)
- define the second window for the last four years (GWINDOW)
- define a viewport in the lower part of the display device (GPORT)
- draw a box around the viewport (GPOLY)
- graph the points and connect them (GPOINT and GDRAW)
- add text (GSCRIPT)
- display the graph (GSHOW)

Stacking Viewports

Viewports can be stacked; that is, a viewport can be defined relative to another viewport so that you have a viewport within a viewport.

A window or a viewport is changed globally through the IML graphics commands: the GWINDOW command for windows, and the GPORT, GPORTSTK, and GPORTPOP commands for viewports. When a window or viewport is defined, it persists across IML graphics commands until another window- or viewport-altering command is encountered. Stacking helps you define a viewport without losing the effect of a previously defined viewport. When a stacked viewport is *popped*, you are placed into the environment of the previous viewport.

Windows and viewports are associated with a particular segment; thus, they automatically become undefined when the segment is closed. A segment is closed whenever IML encounters a GCLOSE command or a GOPEN command. A window or a viewport can also be changed for a single graphics command. Either one can be passed as an argument to a graphics primitive, in which case any graphics output associated with the call is defined in the specified window or viewport. When a viewport is passed as an argument, it is stacked, or defined relative to the current viewport, and *popped* when the graphics command is complete.

For example, suppose you want to create a legend that shows the low and peak points of the data for the ACME stock graph. Use the following statements to create a graphics segment showing this information:

```
call gopen("legend");
call gset('height',5);   /* enlarged to accommodate viewport later */
call gset('font','swiss');
call gscript(5,75,"Stock Peak:  159.5 in 1979");
call gscript(5,65,"Stock Low:   123.6 in 1984");
call gclose;
```

Use the following statements to create a segment that highlights and labels the low and peak points of the data:

```
  /* Highlight and label the low and peak points of the stock */
call gopen("labels");
call gwindow({70 100 87 200}); /* define window */
call gpoint(84,123.625,"circle","red",4) ;
call gtext(84,120,"LOW","red");
call gpoint(79,159.5,"circle","red",4);
call gtext(79,162,"PEAK","red");
call gclose;
```

Next, open a new graphics segment and include the STOCK1 segment created earlier in the chapter, placing the segment in the viewport {10 10 90 90}. Here is the code:

```
call gopen;
call gportstk ({10 10 90 90}); /* viewport for the plot itself */
call ginclude('stocks2');
```

To place the legend in the upper-right corner of this viewport, use the GPORTSTK command instead of the GPORT command to define the legend's viewport relative to the one used for the plot of the stock data, as follows:

```
   call gportstk ({70 70 100 100});   /* viewport for the legend */
   call ginclude("legend");
```

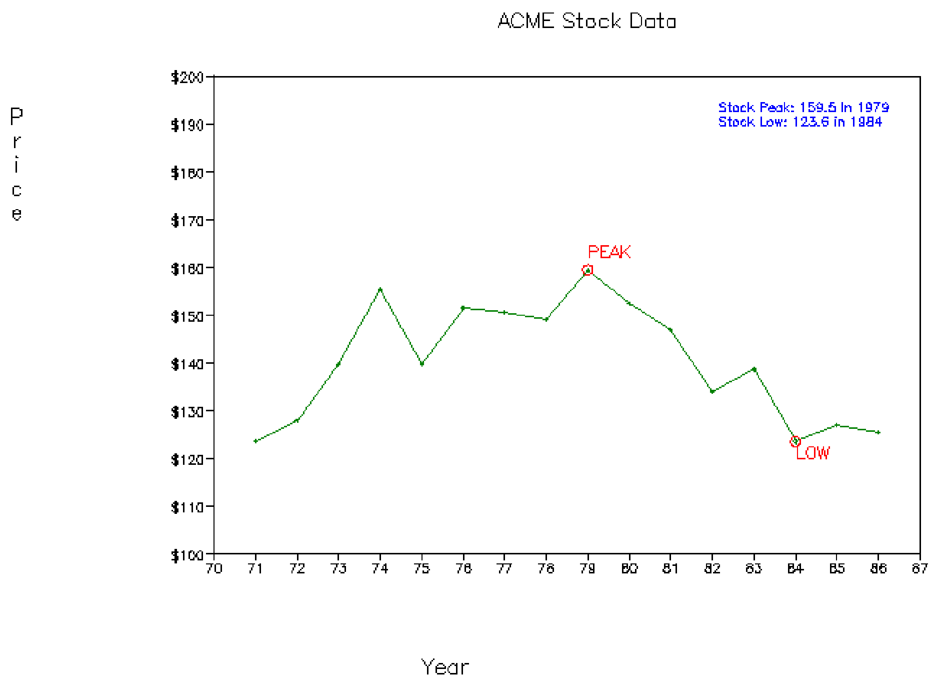
Now pop the legend's viewport to get back to the viewport of the plot itself and include the segment that labels and highlights the low and peak stock points. Here is the code:

```
   call gportpop;   /* viewport for the legend */
   call ginclude ("labels");
```

Finally, display the graph, as follows:

```
   call gshow;
```

Figure 15.8 Stacking Viewports



Clipping Your Graphs

The IML graphics subsystem does not automatically clip the output to the viewport. Thus, it is possible that data are graphed outside the defined viewport. This happens when there are data points lying outside the defined window. For instance, if you specify a window to be a subset of the world, then there will be data lying outside the window and these points will be graphed outside the viewport. This is usually not what you want. To clean up such graphs, you either delete the points you do not want to graph or clip the graph.

There are two ways to clip a graph. You can use the RESET CLIP command, which clips outside a viewport. The CLIP option remains in effect until you submit a RESET NOCLIP command. You can also use the GBLKVP command, which clips either inside or outside a viewport. Use the GBLKVP command to define a blanking area in which nothing can be drawn until the blanking area is released. Use the GBLKVPD command to release the blanking area.

Common Arguments

IML graphics commands are available in the form of call subroutines. They generally take a set of required arguments followed by a set of optional arguments. All graphics primitives take *window* and *viewport* as optional arguments. Some IML graphics commands, like GPOINT or GPIE, accept implicit repetition factors in the argument lists. The GPOINT command places as many markers as there are well-defined (x, y) pairs. The GPIE command draws as many slices as there are well-defined pies. In those cases, some of the attribute matrices can have more than one element, which are used in order. If an attribute list is exhausted before the repetition factor is completed, the last element of the list is used as the attribute for the remaining primitives.

The arguments to the IML graphics commands are positional. Thus, to skip over an optional argument from the middle of a list, you must specify a comma to hold its place. For example, the following command omits the third argument from the argument list:

```
call gpoint(x,y, ,"red");
```

The following list details the arguments commonly used in IML graphics commands:

color	is a character matrix or literal that names a valid color as specified in the GOPTIONS statement. The default color is the first color specified in the COLORS= list in the GOPTIONS statement. If no such list is given, IML uses the first default color for the graphics device. Note that *color* can be specified either as a quoted literal, such as “RED,” a color number, such as 1, or the name of a matrix that contains a reference to a valid color. A color number *n* refers to the *n*th color in the color list. You can change the default color with the GSET command.
font	is a character matrix or quoted literal that specifies a valid font name. The default font is the hardware font, which can be changed by the GSET command unless a viewport is in effect.
height	is a numeric matrix or literal that specifies the character height. The unit of height is the *gunit* of the GOPTIONS statement, when specified; otherwise, the unit is a character cell. The default height is 1 *gunit*, which you can change by using the GSET command.
pattern	is a character matrix or quoted literal that specifies the pattern to fill the interior of a closed curve. You specify a pattern by a coded character string as documented in the V= option in the PATTERN statement (refer to the chapter on the PATTERN statement in *SAS/GRAPH Software: Reference*.

The default pattern set by the IML graphics subsystem is "E," that is, empty. The default pattern can be changed by using the GSET command.

segment-name is a character matrix or quoted literal that specifies a valid SAS name used to identify a graphics segment. The *segment-name* is associated with the graphics segment opened with a GOPEN command. If you do not specify *segment-name*, IML generates default names. For example, to create a graphics segment called PLOTA, use the following statement:

```
call gopen("plota");
```

Graphics segments are not allowed to have the same name as an existing segment. If you try to create a second segment named PLOTA (that is, when the *replace flag* is turned off), then the second segment is named PLOTA1. The *replace* flag is set by the GOPEN command for the segment that is being created. To open a new segment named PLOTA and replace an existing segment with the same name, use the following statement:

```
call gopen("plota",1);
```

If you do not specify a *replace* argument to the GOPEN command, the default is set by the GSTART command for all subsequent segments that are created. By default, the GSTART command sets the *replace* flag to 0, so that new segments do not replace like-named segments.

style is a numeric matrix or literal that specifies an index that corresponds to the line style documented for the SYMBOL statement in the chapter on the SYMBOL statement in *SAS/GRAPH Software: Reference*. The IML graphics subsystem sets the default line style to be 1, a solid line. The default line style can be changed by using the GSET command.

symbol is a character matrix or quoted literal that specifies either a character string that corresponds to a symbol as defined for the V= option of the SYMBOL statement or specifies the corresponding identifying symbol number. STAR is the default symbol used by the IML graphics subsystem.

SAS/IML graphics commands are described in detail in Chapter 23.

Refer also to *SAS/GRAPH Software: Reference* for additional information.

Graphics Examples

This section provides the details and code for three examples that involve SAS/IML graphics. The first example shows a 2 × 2 matrix of scatter plots and a 3 × 3 matrix of scatter plots. A matrix of scatter plots is useful when you have several variables that you want to investigate simultaneously rather than in pairs. The second example draws a grid for representing a train schedule, with arrival and departure dates on the horizontal axis and destinations along the vertical axis. The final example plots Fisher's iris data. The following example shows how to plot several graphs on one page.

Example 15.1: Scatter Plot Matrix

With the viewport capability of the IML graphics subroutine, you can arrange several graphs on a page. In this example, multiple graphs are generated from three variables and are displayed in a scatterplot matrix. For each variable, one contour plot is generated with each of the other variables as the dependent variable. For the graphs on the main diagonal, a box-and-whiskers plot is generated for each variable.

This example takes advantage of user-defined IML modules:

BOXWHSKR	computes median and quartiles.
GBXWHSKR	draws box-and-whiskers plots.
CONTOUR	generates confidence ellipses assuming bivariate normal data.
GCONTOUR	draws the confidence ellipses for each pair of variables.
GSCATMAT	produces the $n \times n$ scatter plot matrix, where n is the number of variables.

The code for the five modules and a sample data set follow. The modules produce Figure 15.1.1 and Figure 15.1.2.

```
      /* This program generates a data set and uses iml graphics     */
      /* subsystem to draw a scatterplot matrix.                     */
      /*                                                             */
   data factory;
     input recno prod temp a defect mon;
     datalines;
       1   1.82675    71.124   1.12404   1.79845         2
       2   1.67179   70.9245  0.924523   1.05246         3
       3   2.22397    71.507   1.50696   2.36035         4
       4   2.39049   74.8912   4.89122   1.93917         5
       5   2.45503   73.5338   3.53382    2.0664         6
       6   1.68758   71.6764   1.67642   1.90495         7
       7   1.98233   72.4222   2.42221   1.65469         8
       8   1.17144   74.0884   4.08839   1.91366         9
       9   1.32697   71.7609   1.76087   1.21824        10
      10   1.86376   70.3978  0.397753   1.21775        11
      11   1.25541    74.888   4.88795   1.87875        12
      12   1.17617   73.3528   3.35277   1.15393         1
      13   2.38103   77.1762   7.17619   2.26703         2
      14   1.13669   73.0157   3.01566         1         3
      15   1.01569   70.4645  0.464485         1         4
      16   2.36641   74.1699   4.16991   1.73009         5
      17   2.27131   73.1005   3.10048   1.79657         6
      18   1.80597   72.6299   2.62986    1.8497         7
      19   2.41142   81.1973   11.1973     2.137         8
      20   1.69218   71.4521   1.45212   1.47894         9
      21   1.95271   74.8427    4.8427   1.93493        10
      22   1.28452   76.7901   6.79008   2.09208        11
      23   1.51663   83.4782   13.4782   1.81162        12
      24   1.34177   73.4237   3.42369   1.57054         1
      25   1.4309    70.7504  0.750369   1.22444         2
```

```
26   1.84851   72.9226   2.92256   2.04468         3
27   2.08114   78.4248   8.42476   1.78175         4
28   1.99175   71.0635   1.06346   1.25951         5
29   2.01235   72.2634    2.2634   1.36943         6
30   2.38742   74.2037   4.20372   1.82846         7
31   1.28055   71.2495   1.24953    1.8286         8
32   2.05698   76.0557   6.05571   2.03548         9
33   1.05429    77.721   7.72096   1.57831        10
34   2.15398   70.8861  0.886068    2.1353        11
35   2.46624   70.9682  0.968163   2.26856        12
36   1.4406    73.5243   3.52429   1.72608         1
37   1.71475    71.527   1.52703   1.72932         2
38   1.51423   78.5824    8.5824   1.97685         3
39   2.41538   73.7909   3.79093   2.07129         4
40   2.28402    71.131   1.13101   2.25293         5
41   1.70251   72.3616   2.36156   2.04926         6
42   1.19747   72.3894    2.3894         1         7
43   1.08089   71.1729   1.17288         1         8
44   2.21695   72.5905   2.59049   1.50915         9
45   1.52717   71.1402   1.14023   1.88717        10
46   1.5463    74.6696   4.66958   1.25725        11
47   2.34151        90        20   3.57864        12
48   1.10737   71.1989   1.19893   1.62447         1
49   2.2491    76.6415   6.64147   2.50868         2
50   1.76659   71.7038   1.70377     1.231         3
51   1.25174   76.9657   6.96572   1.99521         4
52   1.81153   73.0722   3.07225   2.15915         5
53   1.72942   71.9639   1.96392   1.86142         6
54   2.17748   78.1207   8.12068   2.54388         7
55   1.29186   77.0589   7.05886   1.82777         8
56   1.92399   72.6126   2.61256   1.32816         9
57   1.38008   70.8872  0.887228   1.37826        10
58   1.96143   73.8529   3.85289   1.87809        11
59   1.61795   74.6957   4.69565   1.65806        12
60   2.02756   75.7877   5.78773   1.72684         1
61   2.41378   75.9826   5.98255   2.76309         2
62   1.41413   71.3419   1.34194   1.75285         3
63   2.31185   72.5469   2.54685   2.27947         4
64   1.94336   71.5592   1.55922   1.96157         5
65   2.094     74.7338   4.73385   2.07885         6
66   1.19458    72.233   2.23301         1         7
67   2.13118   79.1225    9.1225   1.84193         8
68   1.48076   87.0511   17.0511   2.94927         9
69   1.98502   79.0913   9.09131   2.47104        10
70   2.25937   73.8232   3.82322   2.49798        12
71   1.18744   70.6821  0.682067    1.2848         1
72   1.20189   70.7053  0.705311   1.33293         2
73   1.69115   73.9781    3.9781   1.87517         3
74   1.0556   73.2146   3.21459         1          4
75   1.59936   71.4165   1.41653   1.29695         5
76   1.66044   70.7151  0.715145   1.22362         6
77   1.79167   74.8072   4.80722   1.86081         7
78   2.30484   71.5028   1.50285   1.60626         8
79   2.49073   71.5908   1.59084   1.80815         9
```

```
   80   1.32729   70.9077  0.907698   1.12889        10
   81   2.48874   83.0079   13.0079   2.59237        11
   82   2.46786   84.1806   14.1806   3.35518        12
   83   2.12407   73.5826   3.58261   1.98482         1
   84   2.46982   76.6556   6.65559   2.48936         2
   85   1.00777   70.2504  0.250364         1         3
   86   1.93118   73.9276   3.92763   1.84407         4
   87   1.00017   72.6359   2.63594    1.3882         5
   88   1.90622    71.047     1.047    1.7595         6
   89   2.43744    72.321   2.32097   1.67244         7
   90   1.25712        90        20   2.63949         8
   91   1.10811   71.8299   1.82987         1         9
   92   2.25545   71.8849    1.8849   1.94247        10
   93   2.47971   73.4697    3.4697   1.87842        11
   94   1.93378   74.2952    4.2952   1.52478        12
   95   2.17525   73.0547   3.05466   2.23563         1
   96   2.18723   70.8299  0.829929   1.75177         2
   97   1.69984   72.0026   2.00263   1.45564         3
   98   1.12504   70.4229  0.422904   1.06042         4
   99   2.41723   73.7324   3.73238   2.18307         5
;

proc iml;

   /*-- Load graphics --*/
   call gstart;

   /*--------------------*/
   /*-- Define modules --*/
   /*--------------------*/

   /*   Module : compute contours   */
   start contour(c,x,y,npoints,pvalues);

   /*   This routine computes contours for a scatter plot         */
   /*   c returns the contours as consecutive pairs of columns    */
   /*   x and y are the x and y coordinates of the points         */
   /*   npoints is the number of points in a contour              */
   /*   pvalues is a column vector of contour probabilities       */
   /*   the number of contours is controlled by the ncol(pvalue)  */

      xx=x||y;
      n=nrow(x);
   /* Correct for the mean */
      mean=xx[+,]/n;
      xx=xx-mean@j(n,1,1);

   /* Find principal axes of ellipses */
      xx=xx` *xx/n;
      call eigen(v,e,xx);

   /* Set contour levels */
      c=-2*log(1-pvalues);
      a=sqrt(c*v[1]); b=sqrt(c*v[2]);
```

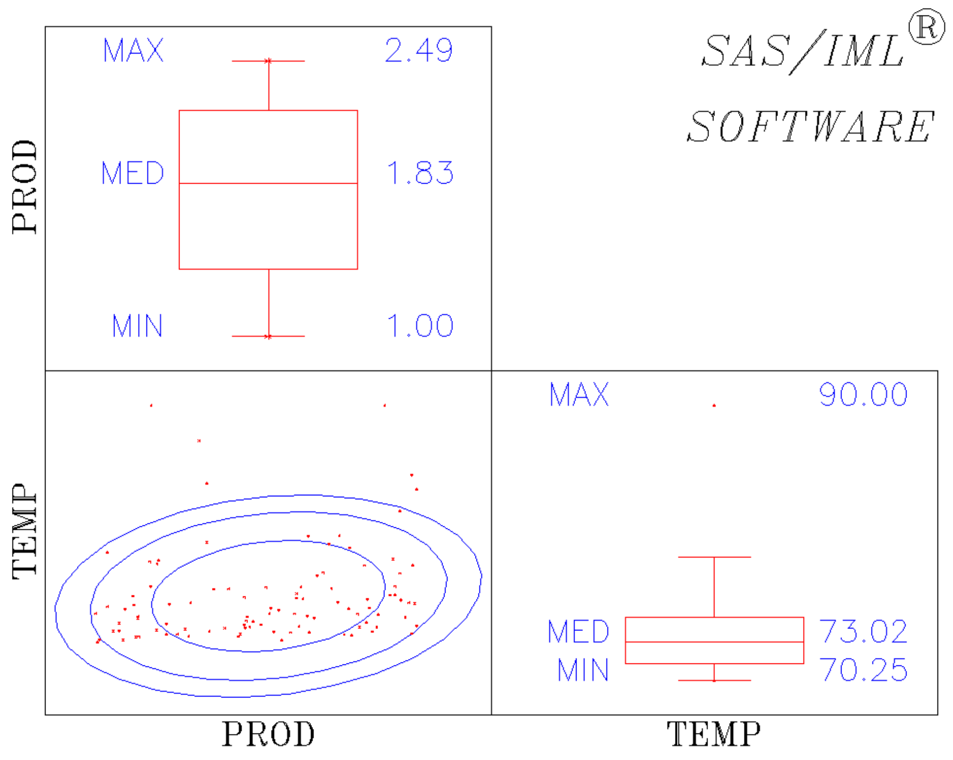
```
/* Parameterize the ellipse by angle */
   t=((1:npoints)-{1})#atan(1)#8/(npoints-1);
   s=sin(t);
   t=cos(t);
   s=s` *a;
   t=t` *b;

/* Form contour points */
   s=((e*(shape(s,1)//shape(t,1)))+mean`@j(1,npoints*ncol(c),1))`;
   c=shape(s,npoints);

/* Returned as ncol pairs of columns for contours */
finish contour;
/*-- Module : draw contour curves --*/
start gcontour(t1, t2);
   run contour(t12, t1, t2, 30, {.5 .8 .9});
   window=(min(t12[,{1 3}],t1)||min(t12[,{2 4}],t2))//
          (max(t12[,{1 3}],t1)||max(t12[,{2 4}],t2));
   call gwindow(window);
   call gdraw(t12[,1],t12[,2],,'blue');
   call gdraw(t12[,3],t12[,4],,'blue');
   call gdraw(t12[,5],t12[,6],,'blue');
   call gpoint(t1,t2,,'red');
finish gcontour;

/*-- Module : find median, quartiles for box and whisker plot --*/
start boxwhskr(x, u, q2, m, q1, l);
   rx=rank(x);
   s=x;
   s[rx,]=x;
   n=nrow(x);

/*-- Median --*/
   m=floor(((n+1)/2)||((n+2)/2));
   m=(s[m,])[+,]/2;

/*-- Compute quartiles --*/
   q1=floor(((n+3)/4)||((n+6)/4));
   q1=(s[q1,])[+,]/2;
   q2=ceil(((3*n+1)/4)||((3*n-2)/4));
   q2=(s[q2,])[+,]/2;
   h=1.5*(q2-q1);   /*-- step=1.5*(interquartile range) --*/
   u=q2+h;
   l=q1-h;
   u=(u>s)[+,];     /*-- adjacent values -----------------*/
   u=s[u,];
   l=(l>s)[+,];
   l=s[l+1,];

finish boxwhskr;

/*-- Box and Whisker plot --*/
start gbxwhskr(t, ht);
```

```
   run boxwhskr(t, up, q2,med, q1, lo);

/*---Adjust screen viewport and data window  */
   y=min(t)//max(t);
   call gwindow({0, 100} || y);
   mid  = 50;
   wlen = 20;

/*-- Add whiskers */
   wstart=mid-(wlen/2);
   from=(wstart||up)//(wstart||lo);
   to=((wstart//wstart)+wlen)||from[,2];

/*-- Add box  */
   len=50;
   wstart=mid-(len/2);
   wstop=wstart+len;
   from=from//(wstart||q2)//(wstart||q1)//
        (wstart||q2)//(wstop||q2);
   to=to//(wstop||q2)//(wstop||q1)//
        (wstart||q1)//(wstop||q1);

/*---Add median line  */
   from=from//(wstart||med);
   to=to//(wstop||med);

/*---Attach whiskers to box  */
   from=from//(mid||up)//(mid||lo);
   to=to//(mid||q2)//(mid||q1);

/*-- Draw box and whiskers  */
   call gdrawl(from, to,,'red');

/*---Add minimum and maximum data points */
   call gpoint(mid, y ,3,'red');

/*---Label min, max, and mean  */
   y=med//y;
   s={'med' 'min' 'max'};
   call gset("font","swiss");
   call gset('height',13);
   call gscript(wstop+ht, y, char(y,5,2),,,,,'blue');
   call gstrlen(len, s);
   call gscript(wstart-len-ht,y,s,,,,,'blue');
   call gset('height');
finish gbxwhskr;

/*-- Module : do scatter plot matrix --*/
start gscatmat(data, vname);
   call gopen('scatter');
   nv=ncol(vname);
   if (nv=1) then nv=nrow(vname);
   cellwid=int(90/nv);
   dist=0.1*cellwid;
```

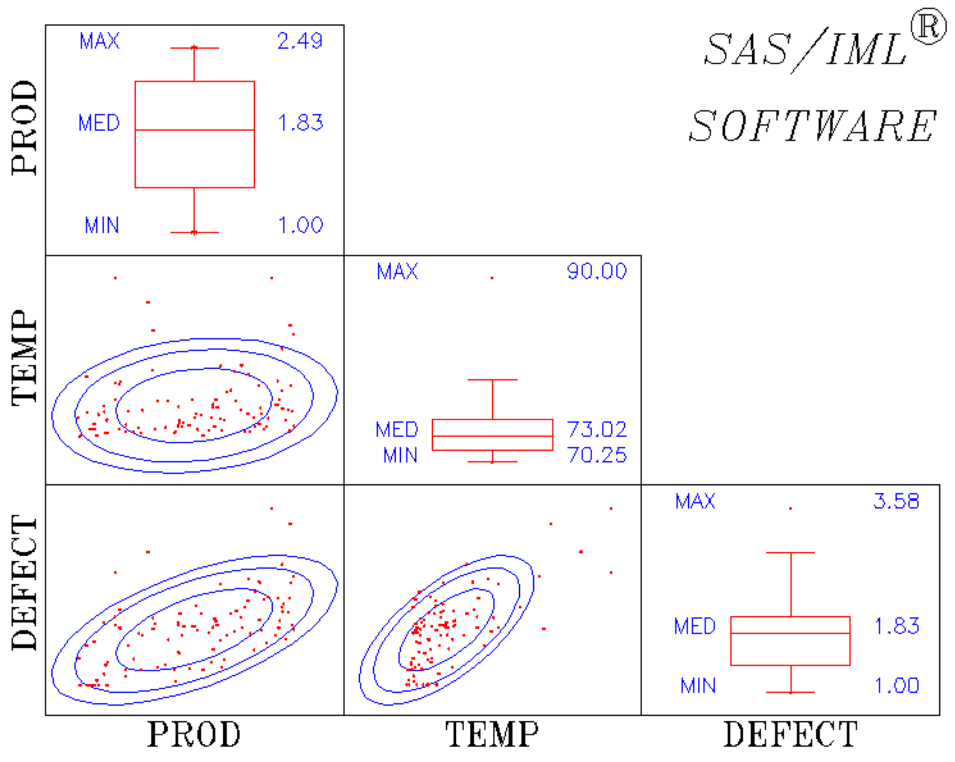
```
      width=cellwid-2*dist;
      xstart=int((90 -cellwid * nv)/2) + 5;
      xgrid=((0:nv)#cellwid + xstart)`;

   /*-- Delineate cells --*/
      cell1=xgrid;
      cell1=cell1||(cell1[nv+1]//cell1[nv+1-(0:nv-1)]);
      cell2=j(nv+1, 1, xstart);
      cell2=cell1[,1]||cell2;
      call gdrawl(cell1, cell2);
      call gdrawl(cell1[,{2 1}], cell2[,{2 1}]);
      xstart = xstart + dist;  ystart = xgrid[nv] + dist;

   /*-- Label variables ---*/
      call gset("height", 5);
      call gset("font","swiss");
      call gstrlen(len, vname);
      where=xgrid[1:nv] + (cellwid-len)/2;
      call gscript(where, 0, vname) ;
      len=len[nv-(0:nv-1)];
      where=xgrid[1:nv] + (cellwid-len)/2;
      call gscript(4,where, vname[nv - (0:nv-1)],90);

   /*-- First viewport --*/
      vp=(xstart || ystart)//((xstart || ystart) + width) ;

   /*  Since the characters are scaled to the viewport      */
   /*   (which is inversely porportional to the             */
   /*   number of variables),                               */
   /*   enlarge it proportional to the number of variables  */

      ht=2*nv;
      call gset("height", ht);
      do i=1 to nv;
         do j=1 to i;
            call gportstk(vp);
            if (i=j) then run gbxwhskr(data[,i], ht);
            else run gcontour(data[,j], data[,i]);
      /*-- onto the next viewport --*/
            vp[,1] = vp[,1] + cellwid;
            call gportpop;
         end;
         vp=(xstart // xstart + width) || (vp[,2] - cellwid);
      end;
      call gshow;
finish gscatmat;

   /*-- Placement of text is based on the character height.       */
   /* The IML modules defined here assume percent as the unit of  */
   /* character height for device independent control.            */
goptions gunit=pct;

use factory;
vname={prod, temp, defect};
```

```
read all var vname into xyz;
run gscatmat(xyz, vname[1:2]);   /*-- 2 x 2 scatter plot matrix --*/
run gscatmat(xyz, vname);        /*-- 3 x 3 scatter plot matrix --*/
quit;

goptions gunit=cell;             /*-- reset back to default --*/
```

Output 15.1.1 2 × 2 Scatter Plot Matrix



Output 15.1.2 3 × 3 Scatter Plot Matrix

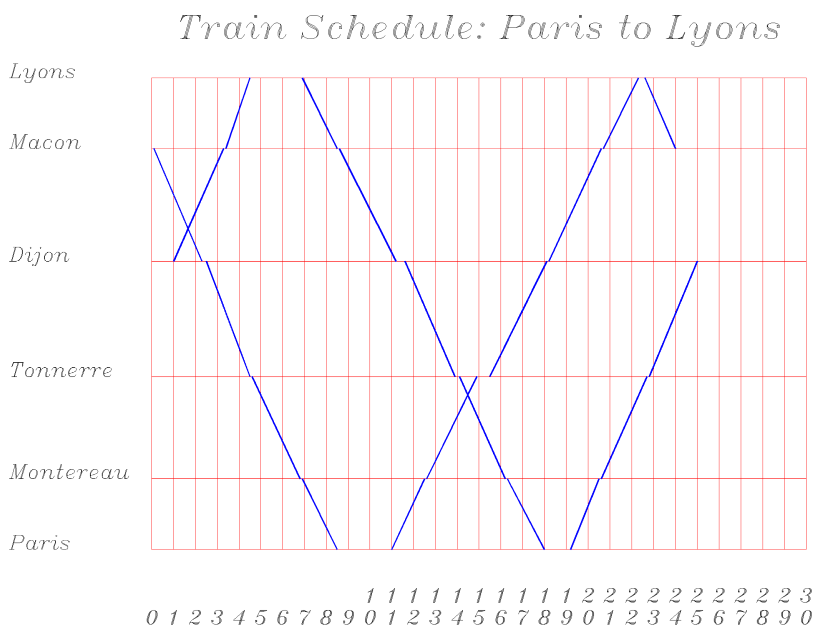


Example 15.2: Train Schedule

This example draws a grid on which the horizontal dimension gives the arrival/departure data and the vertical dimension gives the destination. The first section of the code defines the matrices used. The following section generates the graph. The following example code shows some applications of the GGRID, GDRAWL, GSTRLEN, and GSCRIPT subroutines. This code produces Figure 15.2.1.

```
proc iml;
/*  Placement of text is based on the character height.     */
/*  The graphics segment defined here assumes percent as the */
/*  unit of character height for device independent control. */
   goptions gunit=pct;

   call gstart;
   /* Define several necessary matrices  */
   cityloc={0 27 66 110 153 180}`;
   cityname={"Paris" "Montereau" "Tonnerre" "Dijon" "Macon" "Lyons"};
   timeloc=0:30;
   timename=char(timeloc,2,0);
   /* Define a data matrix  */
   schedule=
      /* origin dest start  end       comment */
         { 1      2   11.0  12.5,   /* train 1 */
           2      3   12.6  14.9,
           3     4   15.5  18.1,
           4     5   18.2  20.6,
           5     6   20.7  22.3,
           6     5   22.6  24.0,
           5     4    0.1   2.3,
           4     3    2.5   4.5,
           3     2    4.6   6.8,
           2     1    6.9   8.5,
           1     2   19.2  20.5,   /* train 2 */
           2     3   20.6  22.7,
           3     4   22.8  25.0,
           4     5    1.0   3.3,
           5     6    3.4   4.5,
           6     5    6.9   8.5,
           5     4    8.6  11.2,
           4     3   11.6  13.9,
           3     2   14.1  16.2,
           2     1   16.3  18.0
         };

   xy1=schedule[,3]||cityloc[schedule[,1]];
   xy2=schedule[,4]||cityloc[schedule[,2]];

   call gopen;
   call gwindow({-8 -35, 36 240});
   call ggrid(timeloc,cityloc,1,"red");
   call gdrawl(xy1,xy2,,"blue");

   /*-- center title -- */
   s = "Train Schedule: Paris to Lyons";
   call gstrlen(m, s,5,"titalic");
   call gscript(15-m/2,185,s,,,5,"titalic");

   /*-- find max graphics text length of cityname --*/
   call gset("height",3);
   call gset("font","italic");
   call gstrlen(len, cityname);
   m = max(len) +1.0
   call gscript(-m, cityloc,cityname);
   call gscript(timeloc - .5,-12,timename,-90,90);
   call gshow;

quit;
goptions gunit=cell;            /*-- reset back to default --*/
```

Output 15.2.1 Train Schedule

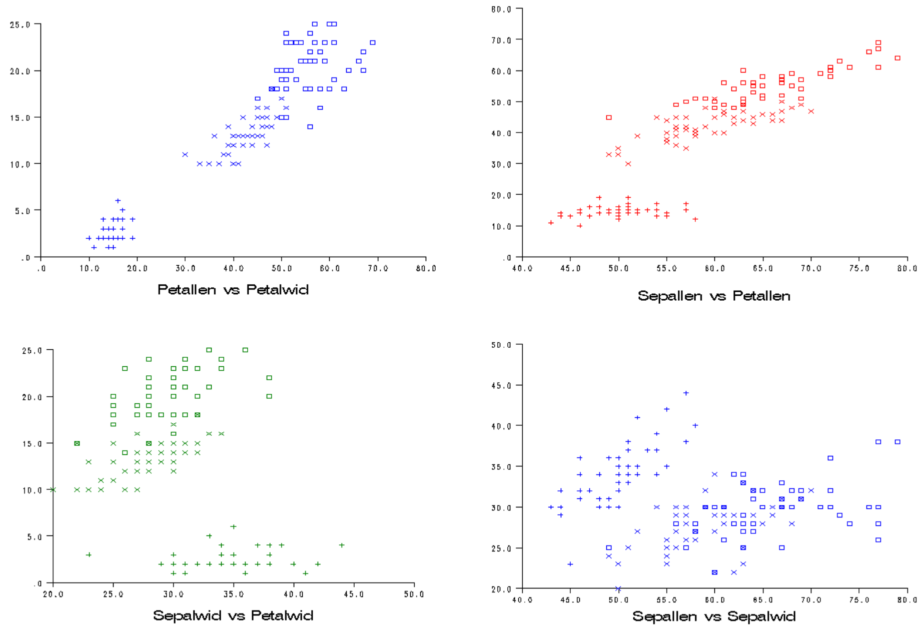


Example 15.3: Fisher's Iris Data

This example generates four scatter plots and prints them on a single page. Scatter plots of sepal length versus petal length, sepal width versus petal width, sepal length versus sepal width, and petal length versus petal width are generated. The following code produces Figure 15.3.1.

```
data iris;
   title 'Fisher (1936) Iris Data';
   input sepallen sepalwid petallen petalwid spec_no @@;
   if spec_no=1 then species='setosa    ';
   if spec_no=2 then species='versicolor';
   if spec_no=3 then species='virginica ';
   label sepallen='sepal length in mm.'
         sepalwid='sepal width  in mm.'
         petallen='petal length in mm.'
         petalwid='petal width  in mm.';
   datalines;

 50 33 14 02 1 64 28 56 22 3 65 28 46 15 2
 67 31 56 24 3 63 28 51 15 3 46 34 14 03 1
 69 31 51 23 3 62 22 45 15 2 59 32 48 18 2
 46 36 10 02 1 61 30 46 14 2 60 27 51 16 2
 65 30 52 20 3 56 25 39 11 2 65 30 55 18 3
 58 27 51 19 3 68 32 59 23 3 51 33 17 05 1
 57 28 45 13 2 62 34 54 23 3 77 38 67 22 3
 63 33 47 16 2 67 33 57 25 3 76 30 66 21 3
 49 25 45 17 3 55 35 13 02 1 67 30 52 23 3
 70 32 47 14 2 64 32 45 15 2 61 28 40 13 2
 48 31 16 02 1 59 30 51 18 3 55 24 38 11 2
 63 25 50 19 3 64 32 53 23 3 52 34 14 02 1
 49 36 14 01 1 54 30 45 15 2 79 38 64 20 3
 44 32 13 02 1 67 33 57 21 3 50 35 16 06 1
 58 26 40 12 2 44 30 13 02 1 77 28 67 20 3
 63 27 49 18 3 47 32 16 02 1 55 26 44 12 2
 50 23 33 10 2 72 32 60 18 3 48 30 14 03 1
 51 38 16 02 1 61 30 49 18 3 48 34 19 02 1
 50 30 16 02 1 50 32 12 02 1 61 26 56 14 3
 64 28 56 21 3 43 30 11 01 1 58 40 12 02 1
 51 38 19 04 1 67 31 44 14 2 62 28 48 18 3
 49 30 14 02 1 51 35 14 02 1 56 30 45 15 2
 58 27 41 10 2 50 34 16 04 1 46 32 14 02 1
 60 29 45 15 2 57 26 35 10 2 57 44 15 04 1
 50 36 14 02 1 77 30 61 23 3 63 34 56 24 3
 58 27 51 19 3 57 29 42 13 2 72 30 58 16 3
 54 34 15 04 1 52 41 15 01 1 71 30 59 21 3
 64 31 55 18 3 60 30 48 18 3 63 29 56 18 3
 49 24 33 10 2 56 27 42 13 2 57 30 42 12 2
 55 42 14 02 1 49 31 15 02 1 77 26 69 23 3
 60 22 50 15 3 54 39 17 04 1 66 29 46 13 2
 52 27 39 14 2 60 34 45 16 2 50 34 15 02 1
 44 29 14 02 1 50 20 35 10 2 55 24 37 10 2
 58 27 39 12 2 47 32 13 02 1 46 31 15 02 1
 69 32 57 23 3 62 29 43 13 2 74 28 61 19 3
 59 30 42 15 2 51 34 15 02 1 50 35 13 03 1
 56 28 49 20 3 60 22 40 10 2 73 29 63 18 3
 67 25 58 18 3 49 31 15 01 1 67 31 47 15 2
 63 23 44 13 2 54 37 15 02 1 56 30 41 13 2
 63 25 49 15 2 61 28 47 12 2 64 29 43 13 2
 51 25 30 11 2 57 28 41 13 2 65 30 58 22 3
 69 31 54 21 3 54 39 13 04 1 51 35 14 03 1
 72 36 61 25 3 65 32 51 20 3 61 29 47 14 2
 56 29 36 13 2 69 31 49 15 2 64 27 53 19 3
 68 30 55 21 3 55 25 40 13 2 48 34 16 02 1
 48 30 14 01 1 45 23 13 03 1 57 25 50 20 3
 57 38 17 03 1 51 38 15 03 1 55 23 40 13 2
 66 30 44 14 2 68 28 48 14 2 54 34 17 02 1
 51 37 15 04 1 52 35 15 02 1 58 28 51 24 3
 67 30 50 17 2 63 33 60 25 3 53 37 15 02 1
;

proc iml;

use iris; read all;

   /*-----------------------------------------------------  */
   /*  Create 5 graphs, PETAL, SEPAL, SPWID, SPLEN, and ALL4 */
   /*   After the graphs are created, to see any one, type   */
   /*                 CALL GSHOW("name");                    */
   /*   where name is the name of any one of the 5 graphs    */
   /* ---------------------------------------------------  */

call gstart;                 /*-- always start with GSTART --*/

   /*-- Spec_no is used as marker index, change 3 to 4 */
   /*-- 1 is + , 2 is x, 3 is *, 4 is a square -------------*/
do i=1 to 150;
   if (spec_no[i] = 3) then spec_no[i] = 4;
end;

   /*-- Creates 4 x-y plots stored in 4 different segments */

   /*-- Creates a segment called petal, petallen by petalwid --*/
call gopen("petal");
   wp = { -10 -5, 90 30};
   call gwindow(wp);
   call gxaxis({0 0}, 75, 6,,,'5.1');
   call gyaxis({0 0}, 25, 5,,,'5.1');
   call gpoint(petallen, petalwid, spec_no, 'blue');
   labs = "Petallen vs Petalwid";
   call gstrlen(len, labs,2, 'swiss');
   call gscript(40-len/2,-4,labs,,,2,'swiss');

   /*-- Creates a segment called sepal, sepallen by sepalwid --*/
call gopen("sepal");
   ws = {35 15 85 55};
   call gwindow(ws);
   call gxaxis({40 20}, 40, 9, , ,'5.1');
   call gyaxis({40 20}, 28, 7, , ,'5.1');
   call gpoint(sepallen, sepalwid, spec_no, 'blue');
   labs = "Sepallen vs Sepalwid";
   call gstrlen(len, labs,2, 'swiss');
   call gscript(60-len/2,16,labs,,,2,'swiss');

   /*-- Creates a segment called spwid, petalwid by sepalwid --*/
call gopen("spwid");
   wspwid = { 15 -5 55 30};
   call gwindow(wspwid);
   call gxaxis({20 0}, 28, 7,,,'5.1');
   call gyaxis({20 0}, 25, 5,,,'5.1');
   call gpoint(sepalwid, petalwid, spec_no, 'green');
   labs = "Sepalwid vs Petalwid";
   call gstrlen(len, labs,2,'swiss');
   call gscript(35-len/2,-4,labs,,,2,'swiss');

   /*-- Creates a segment called splen, petallen by sepallen --*/
call gopen("splen");
   wsplen = {35 -15 85 90};
   call gwindow(wsplen);
   call gxaxis({40 0}, 40, 9,,,'5.1');
   call gyaxis({40 0}, 75, 6,,,'5.1');
   call gpoint(sepallen, petallen, spec_no, 'red');
   labs = "Sepallen vs Petallen";
   call gstrlen(len, labs,2,'swiss');
   call gscript(60-len/2,-14,labs,,,2,'swiss');

   /*-- Create a new segment */
call gopen("all4");
   call gport({50 0, 100 50});  /* change viewport, lower right ----*/
   call ginclude("sepal");      /* include sepal in this graph -----*/
   call gport({0 50, 50 100});  /* change the viewport, upper left  */
   call ginclude("petal");      /* include petal -------------------*/
   call gport({0 0, 50 50});    /* change the viewport, lower left  */
   call ginclude("spwid");      /* include spwid -------------------*/
   call gport({50 50, 100 100});/* change the viewport, upper right */
   call ginclude("splen");      /* include splen -------------------*/

call gshow("all4");
```

Output 15.3.1 Petal Length versus Petal Width


Petallen vs Petalwid
Sepallen vs Petallen
Sepalwid vs Petalwid
Sepallen vs Sepalwid