**INTRODUCTION**

SAS/GRAF provides a creative working environment in which to produce high quality graphical displays. Not only does the SAS system contain many internal options but the annotate facility extends the capability by allowing the user to customize graphs. This paper presents programming tools that have been developed to create specialized graphs in a production environment.

A production environment is characterized by the generation of a volume of output that is highly specific and must be accomplished by a predetermed delivery date. In contrast to the programming style used to produce graphs singularly, a production environment requires a ‘hands off’ programming approach in which the code automatically adapts to database changes with minimal intervention. This adaptability is important because during the production of the volume of work, the graphs are subject to an iterative process of internal and external review. These reviews result in changes and amendments made to some or all of the output prior to final production. The code must be adaptable so that these revisions can be done with minimal time expenditure. For example, if a graph uses note statements or annotation to place text within the procedure area, then any change that affects the size of this area, such as the addition or deletion of a title or footnote, will cause the text to become displaced. Re-estimation of the text placement and editing the appropriate code would then be necessary. A more flexible approach would be to have code that internally determines where text should be placed. In addition, although each project must be tailored to a specific client needs, using a programming approach that is generalizable will save time for all projects.

A production environment, therefore, requires a programming approach that is adaptable, reusable, and efficient. This paper describes several macros and programming strategies that were used to facilitate the production of specialized graphs using the SAS software.

**MACROS**

**SETTING MARGINS WITH %MARGINS**

Most clients have guidelines that the graph margins must adhere to some specified size. The use of null titles and footnotes can be used to create margins of certain sizes. For example,

```
title1 h=2.5in'';
```

will create a 3.0 inch top margin (2.5 inches from the title plus 0.5 inches for the default SAS border for the graphics device). While this method works for many cases, it is not generic and changes must be made to the code if the number of titles or footnotes is varied.

The macro, %MARGINS, solves this problem by creating a full page template that adheres to specified margins. As arguments, this macro takes the four desired margin sizes in inches and the orientation of the graph. The macro converts these inches to percentages and creates a template that has the desired margins after accounting for the default SAS border. Within a program, this macro is called as the final step of production.

The code for %MARGINS is presented below. Several aspects should be noted. First, this macro was written for use with device drivers which, by default, define a 8x10 inch graphics area with a 0.25 inch
margin on two sides and a 0.5 inch margin on the other two sides. However, the paper feeders in our printers are shifted slightly so that the side margins are not exactly the default sizes. Therefore, the code below does not use the exact values of 0.5 and 0.25 inches. In addition, the macro creates a template that is rotated 180 degrees so that the paper holes are on the proper side of the page. An alternative to rotating the template in the definition of the coordinates is to use the 'rotate 180;' option at the end of the template definition. Rotating the paper in the paper feeder is not an option in a production environment.

```latex
\%MACRO MARGINS ( 
  r=, /*size of right margin*/
  l=, /*size of left margin*/
  t=, /*size of top margin*/
  b=, /*size of bottom margin*/
  orient= /*orientation*/
); 
data null;
%If %upcase('orient')="LAND" %then %do;
  tmp_l=100-((&l-0.5)/10)*100;
  tmp_r=((&r-0.4375)/10)*100;
  tmp_b=100-((&b-0.3125)/8)*100;
  tmp_t=((&t-0.25)/8)*100;
%end;
%else %do;
  tmp_l=100-((&l-0.4375)/10)*100;
  tmp_r=((&r-0.5)/10)*100;
  tmp_b=100-((&b-0.3125)/8)*100;
  tmp_t=((&t-0.25)/8)*100;
%end;
  call symput('llx,tmp_l');
  call symput('lly,tmp_b');
  call symput('ulx,tmp_l');
  call symput('uly,tmp_t');
  call symput('urx,tmp_r');
  call symput('ury,tmp_t');
  call symput('urx,tmp_r');
  call symput('ury,tmp_t');
run;
proc gplay igout=work.gseg tc=margins nofs;
tdef margins;
1/llx=&llx lly=&lly
  ulx=&ulx uly=&uly
  urx=&urx ury=&ury
  lrx=&lrx lry=&lry;
run; quit;
\%MEND MARGINS;
```

After \%MARGINS is called within a program, a second proc gplay is needed to replay the desired graph through the newly created template.

**CREATING AN AXIS GAP WITH \%AXIS_GAP**

In many types of studies, there may exist a gap in which no data was collected. For example, suppose data is collected over time with an initial phase of frequent collection, such as every week, followed by six months of no collection and then a resumption of weekly collections. If a bivariate graph is desired with time on the x-axis and some response on the y-axis, using the real time data would result in a large blank space in the middle of the graph. Ideally, the graph would present the data for the first set of collections, have a break in the axis to denote the passage of time and then present the last set of collections.

Using two templates with two calls to PROC G PLOT would be one way to produce such a display. However, this method requires extra programming and does not ensure that the graphs within each template will have the same interval scale unless the size of the templates is calculated relative to the time span represented in each panel.

A solution to this problem is to use the color ‘white’ to literally white out a part of the axis. The macro, \%AXIS_GAP, accomplishes this task. The macro takes as arguments the axis to break, the two values between which the break should occur, whether lines should be added to the axis to signify the break, whether the white out should extend into the area between the x-y axes, and the name of the output annotate data set.

Using this approach does require that the values of the data in the second part of the graph be transformed so that they will plot in sequence immediately after the break. For example, suppose data exists for weeks 1-12 and for weeks 40-52 with the variable ‘week’ containing numeric values
from 1 to 52. In order to plot these data along the same axis with a break between week 12 and 40, the data for weeks 40-52 must be transformed to values of 16-28 if a gap of two intervals is used. The break in the axis would then be specified to occur between values 13 and 15 and an order statement on the axis definition would specify a range from 1 to 28. A format or axis re-labeling with the values= option would also need to be used so that the true value of the weeks 40-52 were indicated below the tick marks.

The code for %AXIS_GAP is presented below:

```
%MACRO AXIS_GAP(
    axis=, /*which axis to break*/
    from=, /*value to start break */
    to=, /*value to end break */
    markers=,/*include break lines*/
    clear=, /*extend white into x-y area*/
    dsn_out= /*name of annotate data set*/
);
%annomac;
data &dsn_out;
%ddanno;
%sequence(after);
if %upcase("&axis")="X" then do;
    %system(2,1);
    %move(&from,0);
    %system(2,9);
    %if %upcase("&clear")="Y" then do;
        %bar(&from,-1,&to,+60,white,3,solid);
    end;
else do;
    %bar(&from,-1,&to,+2,white,3,solid);
end;
else if %upcase("&axis")="Y" then do;
    %system(1,2);
    %move(0,&from);
    %system(9,2);
    %if %upcase("&markers")="Y" then do;
        %bar(-1,&from,+2,&to,white,3,solid);
    end;
end;
if %upcase("&markers")="Y" then do;
    %system(1,2);
    %move(0,&from);
    %system(9,2);
    %line(-1.5,&from,+4,&from,black,1,2);
    %line(0,&to,-3.5,&to,black,1,2);
end;
run;
%MEND AXIS_GAP;
```

**DETERMINING MINIMUM AND MAXIMUM AXIS VALUES WITH %MINMAX**

A common problem in the production of a series of graphs is to make the x and/or y axis uniform across a series of conditions. In a production environment, it is unreasonable to first obtain minimum and maximum values and hard code the axes order. Not only does this require 'hands on' programming but every time the data are changed, the axes orders must be rechecked. In addition, something other than the minimum and maximum of the distribution may be desired, such as the minimum and maximum values for a box and whisker plot or the values for the mean plus and minus a standard error bar. To help solve the problem of setting axes orders, the %MINMAX macro was written. This macro takes the following arguments:

```
%MACRO MINMAX(
    dsn=, /* data set to calculate minimum and maximum values*/
    by=, /* any by variables across which min and max values will be determined*/
    type=, /* type of graph that will be produced */
    values=, /*values are: std (mean +/- standard deviation),
               ste (mean +/- standard error),
               error (the number of standard deviations or standard errors to plot),
               box (box plot) with sub options of: whisker (to specify that whiskers are maximum and minimum values displayed) or all (to specify that all values outside range of whiskers will be displayed),
    stop_0=, /* Y/N to indicate that the axis should start
```
at 0 regardless of minimum value and/or to specify that axis should stop at 0 even if minimum value is below 0*/
var1=, /* name for first plot variable */
var2=, /* name for second plot variable */

The macro determines the minimum and maximum values, rounds them to appropriate values, and sets the increment for the axis. To avoid having axes that cannot be fit within the space allowed in the graphics area, the rounding and increment determination is done so that there are at most 12-15 increments per axis. Because of its length, this macro is not included in the paper but previous SUGI authors have addressed this problem and provided solutions (see references). The %MINMAX macro incorporates many of these solutions and was written to be more general by allowing a variety of graphical displays.

ADDING ADDITIONAL TICK MARKS WITH %ADD_TICK

There are many situations in which it may be desired to have axis tick marks that are not regularly spaced. For example, data may be collected over time but not at regular intervals. Perhaps some data was collected four times over the first day and then once a day after that. In this case, it may be desired to have tick marks just where there is data rather than having regularly spaced tick marks over the entire time span.

Currently, there is no option within SAS/GRAF to remove or add tick marks at any place other than those specified in the ORDER= statement. The %ADD_TICK macro accomplishes this task by placing tick marks at specified locations. As arguments, %ADD_TICK accepts a string of numerical values where the ticks should be placed. A second set of arguments can be used to specify text values and their attributes to display below each tick mark. The macro produces an annotate data set which would be used with the graphic procedure.

The %ADD_TICK macro is presented below:

%MACRO ADD_TICK (  
axis=, /*axis to add ticks to*/  
values=, /*values where ticks will appear*/  
lables=, /*labels to add below tick marks*/  
size=, /*size of tick labels*/  
font=, /*font for tick labels*/  
ds _out= /*name of annotate data set*/
);
%annomac;  
data &ds _out;  
retain t 0;  
dclanno;  
array txt {99} $ _temporary_ (&labels);  
do v= &values;  
t+1;
%sequence(after);  
%system(1,1);  
%move(0,0);  
if %upcase("&axis")="X" then do;
   %system(2,9);
end;
else do; %system(9,2); end;  
%line(v,0,v,-1.5,black,1,0.5);  
%system(1,1);  
%move(0,0);  
%cnt2txt;  
if %upcase("&axis")="X" then do;
   %system(2,9);
end;
else do; %system(9,2); end;
%label(v,2,txt{1},*,0,0,&size,&font,5);  
end;
run;
%MEND ADD_TICK;

To use this macro, all tick marks must first be suppressed with the axis statements of minor=none and major=none. If descriptions for the tick marks are added by the macro then the VALUE=none statement must be used. If the macro does not add the tick mark description then the VALUE= statement must be written so that there are blank values where there are no tick marks. In addition, if the VALUE= statement is used, a blank line must be specified as the first line of the tick mark description. This is necessary since SAS will automatically adjust the tick mark description upwards toward the axis when the tick marks are suppressed with minor and major=none. Failure to add
a blank line as part of the text description will cause the text to be written too close to the axis line.

For example, consider that there is data for days 1 to 3 and then once a week for 2 weeks. In the database, suppose these times are recorded in the variable 'day' with values 1, 2, 3, 10, and 17. The two options for using the macro are as follows:

1) %ADD_TICK

\( \text{values} = \%\text{str}(1,2,3,10,17), \)
\( \text{text} = \%\text{str}(\text{Day 1}' ' \text{Day 2}' ' \text{Day 3}' ' \text{Week 1}' ' \text{Week 2}') ; \)

with an AXIS statement of:
\( \text{AXIS1 v=none minor=none major=none;} \)

OR

2) %ADD_TICK

\( \text{values} = \%\text{str}(1,2,3,10,17); \)

with an AXIS statement of:
\( \text{AXIS1 minor=none major=none} \)
\( \text{value} = \{t=1 \; j=c' ' \; j=c' \text{Day 1}' \}
\( t=2 \; j=c' ' \; j=c' \text{Day 2}' \)
\( t=3 \; j=c' ' \; j=c' \text{Day 3}' \)
\( t=4' \; t=5' \; \ldots \; t=9' \)
\( t=10 \; j=c' ' \; j=c' \text{Week 1}' \)
\( t=11' \; t=12' \; \ldots \; t=16' \)
\( t=17' \text{Week 2}'' \); \)

Both options would require an ORDER=(1 to 17 by 1) statement in the axis definition.

When calling %ADD_TICK, both the &values and the &labels macro variables must be specified as strings. For the above example, the call would be:
\( \%\text{ADD TICK(values=\%str(1,2,3,10,17),} \)
\( \text{labels=\%str(Day 1' 'Day 2' 'Day 3' 'Week 1' 'Week 2'),} \)
\( \text{size=1.5,font=complex,axis=x);} \)

Note that the string of values have commas between them since the string is used in a DO loop. In contrast, the string of labels are quoted values and do not have commas between them since they are used as initial values of a text array.

The two macros, %ADD_TICK and %AXIS_GAP can be used together to produce graphics unique graphics. Figure 1 is an example. The data were collected at three time periods, hours, months, and then hours again.

Figure 1. Example of Using %ADD_TICK and %AXIS_GAP to Produce a Specialized Graph.

CONCENTRATION OVER TIME

![Graph showing concentration over time with specific data points and labels.](image)
It would be possible to produce this graph using 3 templates but it is difficult to get the intervals of the axes for the two ‘hour’ graphs to be identical so that the scale is the same. In addition, the thinness of the last panel will cause some distortion of text even when adjustment of hsize and vsize is made.

An alternative to the use of templates is to define one x-axis and use the %AXIS_GAP macro to create the illusion of three separate axes. The %ADD_TICK macro is then used to place tick marks at the appropriate location. This method does require transformation of the values of the time data so that they are all on a linear scale.

For example, the data for Figure 1 came from three separate datasets. The first data set had the variable ‘hours’ with values ranging from 0 to 350. The second data set had the variable ‘months’ with values 1 through 5. The third data set had the variable hours with values 0 to 100; all data sets had the variable ‘result’, which is the y-axis variable. To create this graphic, the three data sets were brought together and values of months were mapped to new values of hours of 425,450,475,500,525 and the values of the second set of hours were mapped to new values of 600,650, and 700. The resulting data set, therefore, had the variable ‘hour’ which ranged from 0 to 700. The x-axis ORDER statement was then defined as ORDER=(0 to 700). All tick marks were suppressed with the minor=none and major=none axis options.

To produce the figure the following calls to the macros were given:

```
%AXIS_GAP(axis=x,from=350,to=425,clear=y,
           dsn_out=anno1);
%AXIS_GAP(axis=x,from=550,to=600,clear=y,
           dsn_out=anno2);
%ADD_TICK(axis=x,
          values = %str(0,50,100,150,200,250,
                       300,350,425,450,475,500,600,650,700),
          labels = %str('0' '50' '100' '150' '200' '250' '300' '350' '1' '2' '3' '4' '5' '0' '50' '100'))
```

The gplot statement of PLOT RESULT=HOUR; was used with a symbol statement that joined the values. Since the option ‘clear’ was used with the %AXIS_GAP macro, a break in the joined line was produced. The axis labels were added with annotation using the strategy presented below.

**STRATEGIES**

**AUTOMATICALLY ASSOCIATING TEXT WITH THE AXIS**

It is often desired to have descriptive text placed near the graph axes that is in addition to the axes labels and tick mark values. To add this text, either annotate or note statements can be used. If this text is placed using x and y coordinates that reference some point on the graphic display (such as percentage of display) then anytime changes are made which influence the size of the procedure display these coordinates must be redefined since the position of the graph origin will shift. Possible changes might be the addition/deletion of a titles or footnotes or a change in the size of font. One solution would be to use the ORIGIN= option on the axes definition to prevent the graph from shifting position when text changes are made. While this is a viable option, it can result in unused space on the graphics display and may need to be changed if additional text statements are added.

A more adaptable solution is to use the annotate facility to first locate the origin of the graph and then use a relative coordinate system to place text at the desired location. For example, the following code places text at a location that will always be 6% below the x-axis regardless of where the graph appears on the page.

```
%annomac;
data follow;
%dclanno;
%system(1,1);
%move(0,0);
```
Information Visualization

By moving to the origin of the axis and then shifting to relative coordinates prior to placement of the text will ensure that the text will always 'follow' the axis. The %cntl2txt call is necessary to map the XLAST and YLAST internal variables from the %move to the XLSTT and YLSTT internal variables for use with the LABEL function.

For Figure 1, the first axis label was added with the following statements:

%system(1,1);
%move(0,0);
%cntl2txt;
%system(2,9)
%label(175,-4,'Hours',black,0,0,1.5,complex,5);

This sequence was repeated for the other annotated text and the three annotate data set were combined into one.

FLEX CODE

Suppose it is desired to have different line types for each value of z in a y*x=z plot. The line types and the symbols used for each value of z are specified in 'z' symbol statements. However, if the order or types of the values of z change with each data set used in a serial production, it becomes impossible to generalize symbol statements to achieve desired results for every graph. This is an ideal situation in which to use flexcode to write program lines using the PUT statement.

Suppose, for example, that graphs are being produced for y*x=patient and different line types are desired for males and females patients. However, the number and the gender order of patients in the datasets to be graphed is not known a priori. Flexcode can be used create an text file that has the required number of symbol statements each with the proper line type. This text file is then included within the body of the program and serves to create the symbol statements. The following code is an example in which the flexcode was written as a macro and run whenever necessary:

%MACRO SYMBOLS:
filename symbols "[pathname]symbols.inc";
data _null_; again: set patients end = eof;
file symbols;
do until (eof);
   l+1;
   if sex = M then line = 1;
   else line = 2;
   code = "symbol | |left(put(l,2.))| |" v = dot h = 1
   l = join line = " | |left(put(line,1.))| |";
   put code;
   goto again;
end;
run;
%MEND SYMBOLS;

This macro is then called within the program and followed immediately by the statement:

%include "[pathname]symbols.inc",
which includes the text file containing the symbols statements into the program.

CONCLUSIONS

Working in a production environment provides many challenges for the SAS programmer. Fortunately, the flexibility of the SAS software offset the inflexibility of time constraints imposed by such an environment. An initial expenditure of time creating flexible code is well spent when a graphic display is to be used frequently with many types of data.

REFERENCES


Lovell, R. (1989), "You Too Can Produce
and Customize Complex Graphs Using the Annotate Facility and the GSLIDE Procedure with Release 6.03 of SAS/GRAPH Software",


AUTHOR CONTACT

Susan J. Kenny, PhD
Statistical Programming Systems
Quintiles, Inc
P.O. Box 13979
Research Triangle Park, NC 27709
(919)941-1463
skenny@quintiles.com