PharmaSUG 2014 - Paper CC12

A Macro to Automate Symbol Statements in Line Plots

Deli Wang, Ossining, NY
Chunxia Lin, InVentiv Health Clinical, Ossining, NY

ABSTRACT

When processing PROC GPLOT procedure to generate line plots, SAS programmers may have to write several symbol statements and customize specific colors for different groups. The process might be very annoying when data have multiple groups, or programmers are asked to generate/update tons of plots within tight timelines. So isn't wonderful if the symbols in line plots can be assigned based on the input data and line attributes defined in data step, hence programmers take accurate control of line attributes? This paper is trying to write a small macro to empower the programmers with an easy control of symbol statements. The macro provides several advantages: 1. Accurately assign the line attributes like writing a datastep; 2. No longer need to figure out how many symbol statements needed and which symbol is for which line;3. Can easily modify the macro to fit other plotting needs, program once and use it always.

INTRODUCTION

Line plots are the most frequently requested plots, from generating patient pharmacokinetics (PK) profiles to plotting pharmacodynamics (PD) or biomarker data against study days. Writing a single program is relatively easy using PROC GPLOT. However, the situation will be tough when you have tons of plots to generate within very tight timeline and even worse, you are requested to assign a specific color or a line type to a specific group. Hence, we need to have a macro to take care of all the annoying symbol assignments stuff and make the codes more concise and nicer?

PLOT WITH AND WITHOUT GROUPING VARIABLES

The GPLOT procedure plots the values of two or more variables on a set of coordinate axes (X and Y). The coordinates of each point on the plot correspond to two variable values in an observation of the input data set. The PLOT statement is quite simple; however, many programmers have been challenged by its hard to use options and statements. It is not like other software like R which provides very nice default settings, GPLOT need programmers to set up graph environment using many goptions, options as well as different statements. Without well written plotting tools, using GPLOT to generate plots from scratch is not that easy. In GPLOT procedure, the symbol statement is used to define symbol colors or line types.

The basic syntax of symbol statement is:

SYMBOL<1...255> <COLOR=symbol-color|_style_> <MODE=EXCLUDE | INCLUDE> <REPEAT=number-of-times> <STEP=distance<units>> <appearance-option(s)> <interpolation-option> <SINGULAR=n>;
Below example used sashelp.class to plot hight*age, figure 1 used red color in symbol statement, while figure 2 used blue color.

```
GOPTIONS reset=all;

 *Use SASHELP.CLASS as test data;
 DATA class;
   set sashelp.class;
 RUN;

 *Plot Mean Height vs Age (line in red);
 SYMBOL1 color=red i=STD1TJ;
 PROC GPLOT data=class;
   plot height *age;
 RUN;
 QUIT;

 *Plot Mean Height vs Age (line in blue);
 SYMBOL1 color=blue i=STD1TJ;
 PROC GPLOT data=class;
   plot height *age;
 RUN;
 QUIT;
```

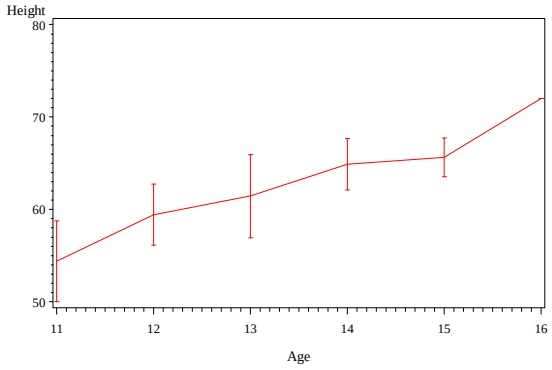

Figure 1. Mean Height vs Age (Line in Red)

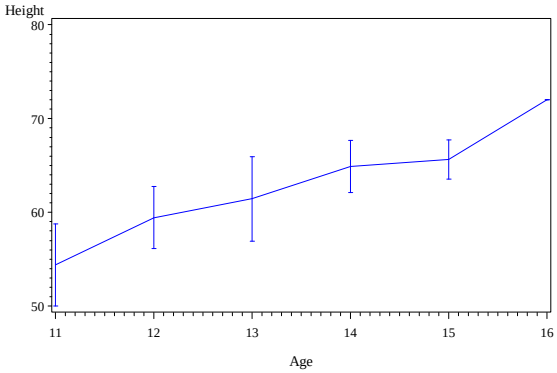

Figure 2. Mean Height vs Age (Line in Blue)

How about adding grouping variable 'sex'? To do that, we have to write 2 symbol statements and also need to figure out which symbol is for which group.

```
SYMBOL1 color=red  i=STD1TJ ;
SYMBOL2 color=blue i=STD1TJ ;

PROC GPLOT data=class;
  plot height *age=sex;
RUN;
QUIT
```

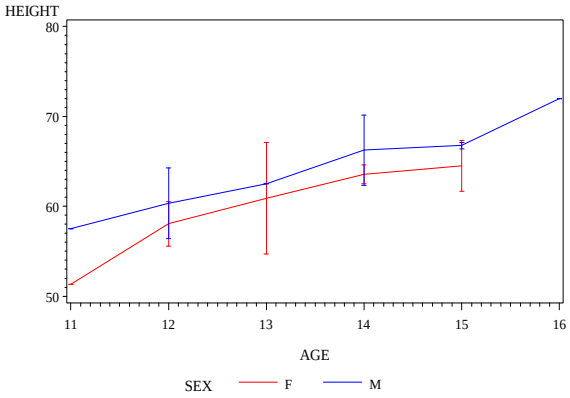

Figure 3. Mean Height vs Age by Sex

In our daily life, the above GPLOT statements are the most common ones. One is y*x, the other one is y*x=z. Z is a grouping variable. In reality, the first syntax is a simplified syntax of y*x=z where z has only one group. So with this in mind, it is not hard to notice that colors (red and blue) were applied to groups by acrobatic order of the values of the grouping variable. In order to streamline the symbol statements, we have to identify two steps:

1) Identify how many groups to assign symbol (n) and which symbol statement is for which line;
2) Automatically write symbol statement based on number of groups.

CALL EXECUTE

CALL EXECUTE is a very useful DATA step call routine which interacts with SAS Macro facility for immediate macro execution during the execution of DATA step. The basic syntax is: CALL EXECUTE (argument). CALL EXECUTE resolves its argument and executes the resolved value at the next step boundary. If argument resolves to macro invocation, the macro executes immediately and DATA step execution pauses while the macro executes. If argument resolves to a SAS statement or if execution of the macro generates SAS statements, the statement(s) execute after the end of the DATA step that contains the CALL EXECUTE routine. CALL EXECUTE can be used for many tasks that would otherwise need many lines of code; it thus helps to create very compact and efficient SAS Codes.

Example:

SASHELP.CLASS dataset contains 19 records, and has 9 records satisfying SEX=”F”.

The following code will print all the data for 9 times since CLASS dataset has 9 records meet the condition (sex=”F”).

```
 DATA _null_;
    set class;
       if sex='F' then call execute('proc print data=class; RUN;');
 RUN;

Log:
NOTE: CALL EXECUTE generated line.
1   + proc print data=class; RUN;
2   + proc print data=class; RUN;
    ...  ...
9  + proc print data=class; RUN;
```

If only printing the data where sex=’F’, then the variable sex has to be added in CALL EXECUTE routine. Below codes will print the data where sex=’F’ 9 times.

```
 DATA _null_;
    set class;
       if sex='F' then call execute("proc print data=class;
        where sex='" ||sex||"' ; RUN;");
 RUN;

Log:
NOTE: CALL EXECUTE generated line.
   1   + proc print data=class; where sex='F' ; RUN;
   2   + proc print data=class; where sex='F' ; RUN;
       ...  ...
   9   + proc print data=class; where sex='F' ; RUN;
```

MACRO WRITITNG

The macro is used to assign symbol statements automatically in GPLOT procedure and customize line attributes based on input dataset in DATA step. The ideas are:

1. First figure out how many distinct values the group variable has, which also decides how many symbol statements needed in PROC GPLOT;
2. Define line attributes based on distinct values of the group variable, which are reflected in a DATA step macro parameter;
3. Use CALL EXECUTE routine to resolve the symbol statements automatically, which helps us to stack up all the symbol statements and run them together.

```
 %MACRO symbol(data =, /* INPUT DATASET*/
               group=, /* GROUP VARIABLE*/
               para =  /* DEFINE LINE ATTRIBUTES IN DATA STEP*/);
 *identifying the number of symbols and also keep all variables in original data. Keep
                         all variables for easy programming;

 PROC SORT data=&data nodupkey out=out;
    by &group;
 RUN;

 DATA out1;
       Set out;
       by &group;
       if first.&group then SEQ+1;
 RUN;
```

```
*Use ‘call execute’ to conditionally process codes while taking data step values to
apply the most often used symbol parameters(c,I,v,l,f,mode,w,h).some parameters were
set with defaults, if &para is defined differently and executed, some parameters in
data step may be overwrriten;

 DATA _null_;
      LENGTH C V I f $100.;
      Set out1;
         w=1;
         h=1;
         i='std1tjm';
         f='SWISS';
         l=SEQ;
         c='_STYLE_';
         v='';
         &para.;
      call execute( "symbol"||strip(put(seq,best.))||"  c='"||strip(c)||
      "'  v='"||strip(v)|| "' L=" ||strip(put(l,best.))||"  i=  " ||STRIP(i)||
      "  font='"||strip(f)||" ' mode=include"||" width="||strip(put(w,best.))||
      "  h="||strip(put(h,best.)) ||" ;"  );
 RUN;
 %MEND symbol;
```

Example call without defining ‘para’:

```
%symbol(data=sashelp.class,group=sex,para=%str(  ));

NOTE: CALL EXECUTE generated line.
1   + symbol1  c='_STYLE_'  v='' L=1  i=  std1tjm  font='SWISS ' mode=include width=1  h=1 ;
2   + symbol2  c='_STYLE_'  v='' L=2  i=  std1tjm  font='SWISS ' mode=include width=1  h=1 ;
```

If leaving ‘para’ blank, the default colors from sas or the predefined values for w,h,l,f and l will be used. This makes it easier for programmers to test program first and then use the raw data to change the line attributes.

Blow codes plugs the macro in PROC GPLOT procedure, and redefined the line attributes in macro variable para which overwritten the predefined line attributes in data step. Figure 4 is the output.

```
 %symbol(data =sashelp.class,

              group=sex,
              para =
 %str(
IF SEX='M' THEN DO;C='RED'; L=1;H=2;W=2;V=' ';F='SWISS';I='STD1TJ';END;
If SEX='F' THEN DO;C='BLUE';L=2;H=1;W=1;V=' ';F='SWISS';I='STD1TJ';END;
  ));

 PROC GPLOT data=class;
    PLOT  height *age=sex;
 RUN;
 QUIT;

Log:

NOTE: CALL EXECUTE generated line.
1    + symbol1  c='BLUE'  v=''  L=2  i=  STD1TJ  font='SWISS ' mode=include width=1  h=1 ;
2    + symbol2  c='RED'   v=''  L=1  i=  STD1TJ  font='SWISS ' mode=include width=2  h=2 ;
```

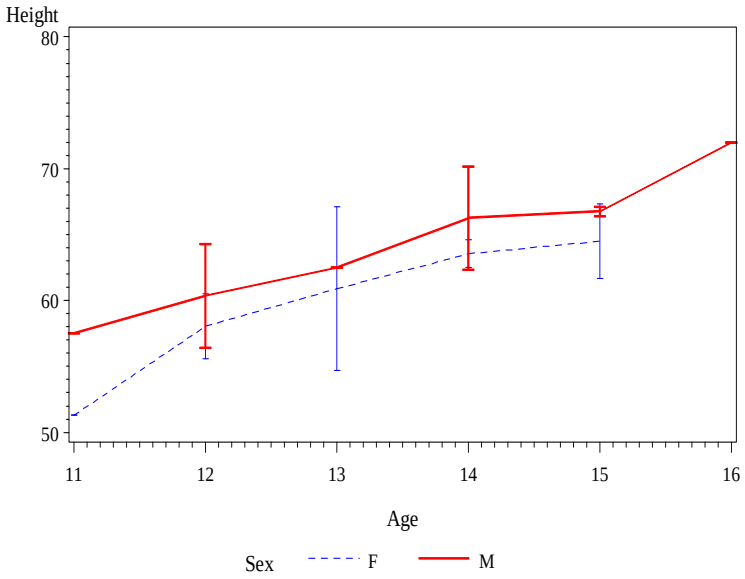

Figure 4. Mean Height vs Age by Sex

The macro has below advantages:

1. In the macro, the line attributes are associated with actual values of the group variable, and defined in a data step macro parameter, hence it is very easy, convenient and efficient to customize the line attributes whenever requested. E.g the customer request to set “SEX=”F” group as red, or orange or line thinner/thicker, or whatever color/style they would like and for whatever group. In common sense, the programmers need go back to PROC GPLOT symbol statements to figure out which color/line stands for which group first, then change attributes per request. The process isn’t fun and can even be a nightmare when having multiple groups and tons of plots to update. However, the job will be a piece of cake using the macro, because the changes are targeted on actual input values, so will be 100% accurate and confident.

2. No need to figure out how many symbol statements needed in PROC GLOT and write them down respectively, because the CALL EXECUTE routine will take care of all those tedious stuff.

3. The small macro can be easily incorporated into PROC GPLOT procedure, can be taken as a starting point reference to develop a fully functional, robust and more customized plotting tool.

TO EXPAND THE MACRO TO 2Y PLOTS

2Y line plots are also very popular in pharmaceutical industry. Figure 5 is the output of 2Y line plots generated by below codes. After studying the outputs we know that GPLOT will assign the first Y based on the order of the first grouping variable and assign the second Y based on the order of the second grouping variable.

```
SYMBOL1 color=red   i=STD1TJ ;
SYMBOL2 color=blue  i=STD1TJ ;
SYMBOL3 color=green i=STD1TJ ;
SYMBOL4 color=gold  i=STD1TJ ;

PROC GPLOT data=class;
  plot  height*age=sex;
  plot2 weight*age=sex;

RUN;
QUIT;
```

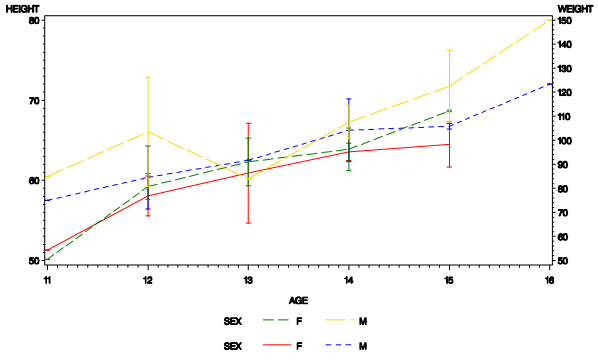

Figure 5. Mean Height and Weight vs Age by Sex

Knowing the order of symbol assignments is the key to our programming, below is the modified Macro to incorporate 2Y plots:

```
%MACRO symbol (data =, /* INPUT DATASET*/
               group1=, /* GROUP VARIABLE1*/
               group2=, /* GROUP VARIABLE2*/
               para  =  /* DEFINE LINE ATTRIBUTES IN DATA STEP*/);

*assign ‘data’ variable in order to make the macro also be applicable
                      to single y plots;

PROC SORT data=&data nodupkey out=out1;
   by &group1;
RUN;

DATA out1;
      Set out1;
      data=1;
RUN;

%if &group2 ^= %then %do;
PROC SORT data=&data nodupkey out=out2;
   by &group2;
RUN;

DATA out2;
      Set out2;
      data=2;
RUN;
%end;

DATA out;
   set out1  %if &group2 ^= %then %do;  out2 %end;   ;
      seq=_n_;
RUN;

*Use ‘call execute’ to conditionally process codes while taking data step values to
apply the most often used symbol parameters(c,I,v,l,f,mode,w,h).some parameters were
set with defaults, if &para is defined differently and executed, some parameters in
data step may be overwrriten;

Data _null_ ;
     LENGTH C V I F $100.;
     Set out;
        w=1;
        h=1;
        i='std1tjm';
        f='SWISS';
        l=SEQ;
        c='_STYLE_';
        v='';
     &para.;
     call execute( "symbol"||strip(put(seq,best.))||"  c='"||strip(c)||
     "'  v='"||strip(v)|| "' L=" ||strip(put(l,best.))||"  i=  " ||STRIP(i)||
     "  font='"||strip(f)||" ' mode=include"||" width="||strip(put(w,best.))||
     "  h="||strip(put(h,best.)) ||" ;"  );
RUN;
%MEND symbol;
```

Test:

```
%symbol(data=sashelp.class,group1=sex,group2=sex,para=%str(
 if data=1 then do;c='black'; w=2; end;
 if data=2 then do;c='red';   w=2; end;));

PROC GPLOT data=sashelp.class;
   plot  height*age=sex;
   plot2 weight*age=sex;
 RUN;
 QUIT;

Log:
NOTE: CALL EXECUTE generated line.
1   + symbol1  c= black  v=   L=1  i=   std1tjm font='SWISS ' mode=include width=2 h=1 ;
2   + symbol2  c= black  v=   L=2  i=   std1tjm font='SWISS ' mode=include width=2 h=1 ;
3   + symbol3  c= red    v=   L=3  i=   std1tjm font='SWISS ' mode=include width=2 h=1 ;
4   + symbol4  c= red    v=   L=4  i=   std1tjm font='SWISS ' mode=include width=2 h=1 ;
```

Here is the output:

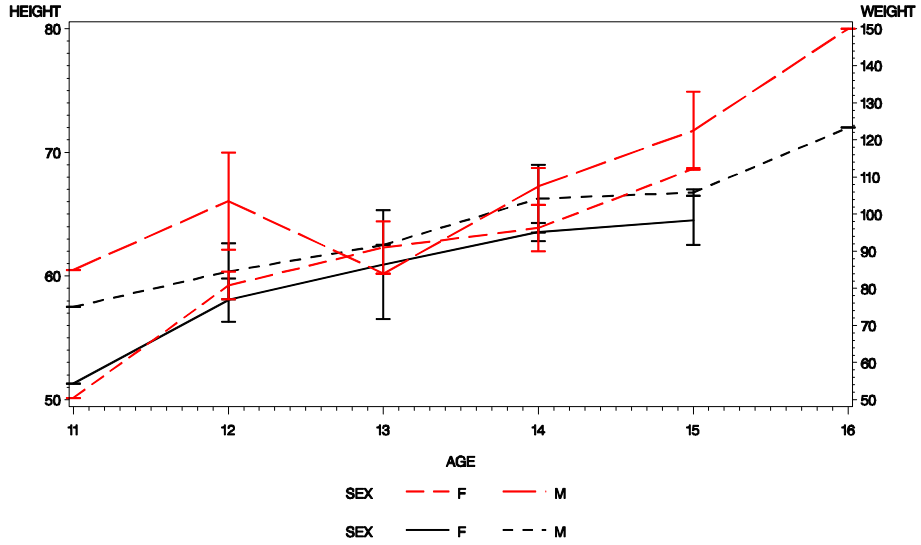

Figure 6. Mean Height and Weight vs Age by Sex

It is worth noting that after execution of the macro, there is a dataset called ‘out’; this data is the most useful source to assign the line attributes.

So far we have discussed and developed the Macro for symbol statement. However, because the limitation of symbol statements, this macro will work only if the total number of distinct values of group1 and group 2 is no more than 255, otherwise we have to use repeat option or use other solutions. Following the macro, we can expand further, to wrap the whole program into a single big macro very easily, the authors will not elaborate the code here, but the macro call could be like this:

```
%symbol (data=sashelp.class, group1=sex, group2=sex, para=%str(some attributes
definition or none), x=age, y1=height,y2=weight);
```

Now, without considering symbol statements, we are able to generate line plots using a single macro call.

CONCLUSION

This paper only touches upon a macro to assign symbol statements to generate line plots. Even though it has some limitations, in most cases, we seldom have a task that needs 255 symbol statements, even some time we do, we can still work around the issue. This paper can also serve as a starting point for programmers who want to generate other kinds of plots using GPLOT; they can always modify the macro to expand the usability to fit their needs.

REFERENCES

1. SAS/BASE Software: Version 9, SAS® Institute Inc., Cary NC
2. SAS/MACRO Software: Version 9.1.3, SAS® Institute Inc., Cary NC

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Deli Wang
deli.wang@hotmail.com

Chunxia Lin
InVentiv Health Clinical
(914) 9230173
lin.trisha@yahoo.com