PharmaSUG 2015 - Paper SP04

Means Comparisons and No Hard Coding of Your Coefficient Vector – It Really Is Possible!

Frank Tedesco, United Biosource Corporation, Blue Bell, Pennsylvania

ABSTRACT

When doing mean comparisons using a linear vector of coefficients, typically we need to modify our SAS PROC statements to account for different study designs, for instance when doing a submission (Phases I, II,III). What if you didn't need to hard code your linear vector in every program specific to the analysis? The number of linear coefficients in the vector are dependent on the number of treatment levels and cohort(s) being analyzed, in other words the levels of your independent variable(s) will change with each study design.

This paper will use pharmacokinetic data as an example with the added difficulty of analyzing drug metabolites which often have parameter values with measurements below quantifiable limits. Often this causes treatment levels and cohorts to drop from the analysis. The goal of this paper is to demonstrate that the use of the macro facility can provide for SAS program stability and reduce SAS program maintenance ultimately resulting in a time and cost savings across studies.

INTRODUCTION

The idea of automating code for MEANS custom hypothesis testing formed while working with the pharmacokinetic (PK) profiles of a cancer drug therapy and metabolites following multiple oral dosages. We were faced with creating comparisons of the pharmacokinetic action between cohorts of individuals with clinically impaired and normal liver function. With tight timelines, we needed to start programming on partial, incomplete data but also needed to ensure the code was robust enough to correctly summarize the full data when available. In addition, we wanted to write code that could be easily reused, generating cost efficiencies across studies and projects.

This application of macro use in SAS procedures that rely on linear contrasts to do multiple comparisons. PK data is referenced here but this could be extrapolated to other types of data but the concept remains the same: automation brings about efficiency, accuracy and time savings. Constant program maintenance and repetitive validation is time consuming, expensive and prone to error. When automation can be applied, programmers have more time to address other critical data issues revealed through data review. Using macros can also help to ensure complete and accurate hypothesis testing and validation.

CHALLENGE

In Phase I clinical trials, statistical analysis of single and multiple dose PK data involves a single analysis of multiple PK parameters and metabolites. Data is usually presented by analyte, treatment, cohort and period. Given that patient PK evaluability requires a specific set of programmatic data handling rules and that patient evaluability may vary from study to study, continual maintenance of programs is required to ensure that the linear contrast statements are constructed appropriately for custom hypothesis testing producing means, confidence intervals and p-values.is the paper body.

The SAS® GLM procedure allows for custom means hypothesis testing using the contrast or estimate statements and a linear vector of coefficients. In order to ensure accuracy with the automation process, it is critical to evaluate the number and order of levels of the CLASS variables and BY variables because the number of levels of CLASS may vary within each value of BY variable. These levels must be determined to accurately construct the linear combination of coefficients in the contrast or estimate vector. An example of the syntax of SAS GLM model for PK data analysis, for simplicity, a one-way analysis of variance is shown:

```
PROC GLM data=pkdata;
     by anord analyte;
     class cohort;
     model &pkparam = cohort;
     lsmeans cohort /pdiff stderr alpha=0.5 CL;
     estimate "mild vs normal" cohort -1 1 0 0;
     estimate "moderate vs normal" cohort -1 0 1 0;
     estimate "severe vs normal" cohort -1 0 0 1;
run;
```

The complication becomes apparent when the same code is to be used in validated programs through the course of the trial for repetitive deliverables (ie: safety monitoring committee meetings, FDA special requests, interim analyses, etc.) where the number of cohort levels may vary due to patient enrollment rates or protocol changes. It is important and most efficient to rerun and reuse code without modification or revalidation throughout the clinical trial. This saves production and validation time, reduces costs and provides accurate results within shorter timelines.

Solution: SAS macro

Before running the macro, analyze the data to get information about the CLASS variable: how the custom hypothesis contrast statement text should appear, the comparator cohort and the order of cohorts. This may also include formatting of the CLASS variable or determining BY variables to use to subset the data. The macro can then use SAS function data lookups to create model syntax. BY variables are used to subset the data instead of using a BY statement since PROC GLM must run uninterrupted. By using BY subsets, the data lookups can be done through iterative runs of PROC GLM producing the required syntax.

The parameters that need to be defined for model syntax are: CLASS, BY, dependent or independent variables and the format for the CLASS variable. Sample code follows:

```
%mlsm (indata=pkad, /* analysis data ie, PK analyte, patient cohort */
       depvar=cmax, /* dependent variable i.e., PK parameter*/
       indvar=cohort, /* independent variable , i.e., cohort*/
       clsvar=cohort, /* class variable with discrete levels ie cohort*/
       stdval=Normal, /* class variable level used as standard cohort comparator*/
       byvar=anord analyte cycledy, /* by variables i.e., PK analyte cycle day */
       anord=anord /* analyte order i.e., 1,2,3...*/
       );


%macro mlsm(indata=,depvar=,indvar=,clsvar=,stdval=,byvar=,anord=);
```

Get data set unique by variable value sort order:

```
%if &byvar ^= %then
%do;

       /* get the number of by variables */
       data _null_;
              array by{*} &byvar;
              byn= dim(by);
              call symput('bylen',byn);
       run;

       %put number of by variables bylen=&bylen;

       proc sort data=&indata out=sortby(keep=&byvar) nodupkey;
              by &byvar;
       run;

       %let bsortid=%sysfunc(open(work.sortby,i)); /* open the sorted data set */
       %let blevels=%sysfunc(attrn(&bsortid,nobs)); /* determine the number of obs */
       %let rc=%sysfunc(close(&bsortid));
       %put number of by var level combinations=&blevels;
```

```
	%if &clsvar ^= %then
	%do;
		proc sort data=&indata out=sortbycl(keep=&byvar &clsvar) nodupkey;
			by &byvar &clsvar;
		run;

		%let bcsortid=%sysfunc(open(work.sortbycl,i));/*open the sorted dataset*/
		%let bclevels=%sysfunc(attrn(&bcsortid,nobs)); /*get number of obs*/
		%let rc=%sysfunc(close(&bcsortid));
		%put number of by and class var level combinations =&bclevels;
		%let byclvar=&byvar &clsvar; /* by and class variable combo */

	%end;
%end;
```

Get data set unique class variable value sort order:

```
%if &clsvar ^= %then
%do;

	proc sort data=&indata out=sortcls(keep=&clsvar) nodupkey;
		by &clsvar;
	run;

	%let csortid=%sysfunc(open(work.sortcls,i)); /*open the sorted dataset*/
	%let clevels=%sysfunc(attrn(&csortid,nobs)); /*get number of obs*/
```

Get class variable varnum and vartype:

```
	%let rc=%sysfunc(fetchobs(&csortid,1));
	%let clsvnm = %sysfunc(varnum(&csortid,&clsvar));

	%if %sysfunc(vartype(&csortid,&clsvnm)) = N %then
	%do;
		%let clsvaln=%sysfunc(getvarn(&csortid,&clsvnm));
		%put clsvnm=&clsvnm clsvaln=&clsvaln;
	%end;
	%if %sysfunc(vartype(&csortid,&clsvnm))= C %then
	%do;
		%let clsvalc=%sysfunc(getvarc(&csortid,&clsvnm));
		%put clsvnm=&clsvnm clsvalc=&clsvalc;
	%end;
```

Get class variable formatted values:

```
	%let vfmt = %sysfunc(varfmt(&csortid,&clsvnm));
	%let vinfmt = %sysfunc(varinfmt(&csortid,&clsvnm));
	%put classvar format=&vfmt;
	%put classvar informat=&vinfmt;
	%let rc=%sysfunc(close(&csortid));
	%put number class var levels = &clevels;
%end;
```

GLM WHERE clause subset:

Start building WHERE clause based on BY variable levels. Use do-loop processing within the macro to repeat the model statements and issuing the appropriate L vector construction of linear coefficients for each iteration of GLM:

```
%if &byvar ^= %then
%do;
       %let bsortid=%sysfunc(open(work.sortby,i)); /* open the sorted data set */
       %do l = 1 %to &blevels;
              %let rc=%sysfunc(fetchobs(&bsortid,&l)); /* fetch each record and read in
                                                          values of the by variables */
```

Build WHERE clause with unique values of each BY variable:

```
              %do b = 1 %to &bylen;
                     %let by = %scan(&byvar,&b);
                     %let vnum=%sysfunc(varnum(&bsortid,&by));
                     %put by variable &by vnum &vnum;
                     %if &b=1 %then
                     %do;
                            %if %sysfunc(vartype(&bsortid,&vnum))= C %then
                            %do;
                                   %let byvalc=%sysfunc(getvarc(&bsortid,&vnum));
                                   %put &by by variable value byvalc &byvalc;

                                   /* where &by='&byvalc' note that if BYVAR is
                                      character, then need quotes around &BYVAL1*/
                                   %let whr=%str(where &by="&byvalc");
                                   %let whr_stg=%str(where &by="&byvalc");
                                   %put first part of where clause &whr;
                            %end;

                            %if %sysfunc(vartype(&bsortid,&vnum))= N %then
                            %do;
                                   %let byvaln=%sysfunc(getvarn(&bsortid,&vnum));
                                   %put &by by variable value byvaln = &byvaln;
                                   /* where &by=&byvaln note that if BYVAR is numeric,
                                      then NO quotes around &BYVAL1 */
                                   %let whr=%str(where &by=&byvaln);
                                   %let whr_stg=%str(where &by=&byvaln);
                                   %put first part of where clause &whr;
                            %end;
              %end; /* end of b to &bylen */


              /* allow for multiple by variables */
              %if &b>1 %then
              %do;
                     %if %sysfunc(vartype(&bsortid,&vnum))=C %then
                     %do;
                            %let byvalc=%sysfunc(getvarc(&bsortid,&vnum));
                            /* and &by='&byvalc' note that if BYVAR is character,
                               need quotes around BYVALC*/

                            %if &b=2 %then
                            %do;
                                   %let whr_stg=%str(&whr and &by="&byvalc");
```

```
                                %put 2nd where piece &whr_stg;
                        %end;
                        %if &b>2 %then
                        %do;
                                %let whr_stg=%str(&whr_stg and &by="&byvalc");
                                %put &b where piece &whr_stg;
                        %end;
                %end;

                %if %sysfunc(vartype(&bsortid,&vnum))=N %then
                %do;
                        %let byvaln=%sysfunc(getvarn(&bsortid,&vnum));
                        /* and &by=&byvaln note that if BYVAR is numeric,
                           NO quotes around BYVALN*/

                        %if &b=2 %then
                        %do;
                                %let whr_stg=%str(&whr and &by=&byvaln);
                                %put 2nd where piece &whr_stg;
                        %end;
                        %if &b>2 %then
                        %do;
                                %let whr_stg=%str(&whr_stg and &by=&byvaln);
                                %put &b where piece &whr_stg;
                        %end;
                %end;
        %end; /*end of if then b>1*/

        %let wher&l=%str(&whr_stg;);
        %put FINAL where clauses for each by variable combination
        wher&l=&&wher&l;

    %end; /*end of by level loop*/

    %let rc=%sysfunc(close(&bsortid));
%end; /*end of byvar loop*/
```

Find all by variable / cohort / treatment / analyte combinations that satisfy analysis and build linear combination of coefficients for the CONTRAST/ESTIMATE statement in model:

```
%if &byvar ^= %then
%do;
    %do l = 1 %to &blevels;
        data lsm&l;
        set &indata;
                &&wher&l
        run ;
```

Get data set unique class variable value sort order:

```
%if &clsvar ^= %then
%do;
    data lsm&l;
    set lsm&l;
        format &clsvar;
    run ;

    proc sort data=lsm&l out=sortcls&l(keep=&clsvar) nodupkey;
        by &clsvar;
    run;
```

Apply CLASS variable formats:

```
%if &vfmt ^= %then
%do;
       data sortcls&l;
       set sortcls&l;
              by &clsvar;
              fmt&clsvar = put(&clsvar,&vfmt);
       run ;
+%end;


/* Determine j*l levels BY variable levels * CLASS variable levels */

%let csortid=%sysfunc(open(work.sortcls&l,i)); /* open the sorted dataset */
%let clevels=%sysfunc(attrn(&csortid,nobs)); /* determine the number of obs */
%let contrast&l=;

%do j=1 %to (&clevels+1);
       %do c = 1 %to &clevels;

       /*get class variable varnum and vartype*/

       %let rc=%sysfunc(fetchobs(&csortid,&c));
       %let clsvnm = %sysfunc(varnum(&csortid,&clsvar));


       %if %sysfunc(vartype(&csortid,&clsvnm))= C %then
       %do;
              %let clsvalc=%sysfunc(getvarc(&csortid,&clsvnm));
              %if &c=1 %then
              %do;
                     %let contrstc = &clsvalc;
              %end;
              %else
              %do;
                     %let contrstc = %str(&contrstc &clsvalc);
              %end;
              /* set position of comparator treatment */
              %if &clsvalc = &stdval %then %let location = &c;
       %end;

       /* Allow for CLASS variables with formats */

       %let fclsvnm = %sysfunc(varnum(&csortid,fmt&clsvar));

       %if %sysfunc(vartype(&csortid,&fclsvnm))= C %then
       %do;
              %let fclsvalc=%sysfunc(getvarc(&csortid,&fclsvnm));
              %if &c=1 %then
              %do;
                     %let fcontrstc = &fclsvalc;
              %end;
              %else
              %do;
                     %let fcontrstc = %str(&fcontrstc &fclsvalc);
              %end;
              %if &fclsvalc = &stdval %then %let location =&c;
       %end;
```

Find position the comparator treatment:

```
       %if &j>&location %then %let inc=&c+1;
       %let val=0;
```

```
       %if &location=&j %then %let val=1;
       %else %if &j=&inc %then %let val=-1;
       %else %if &j< &location and &c=&j %then %let val=-1;
```

Create CONTRAST/ESTIMATE statement for each WHERE clause:

```
       %if &j=1 %then %let contrast&l=&val;
       %else %let contrast&l=%str(&&contrast&l &val);
       %let temp=%str(&stdval vs &fclsvalc);
       %let estmt&l = %str(estimate "&temp" &indvar &&contrast&l);
       %put &&wher&l contrast statement estmt = &&estmt&l;

       %end; /*end c loop*/
  %end; /*end j loop*/

 %end; /*class var loop*/
%end; /*end of byvar loop */
```

Use ODS statements to format and store statistics from of each run of the model. Use MACRO facility to store the WHERE clause and string of L vector of coefficients in MACRO variables associated with each specific iteration of PROC GLM.

```
ods output means =n&num;
ods output overallanova=over&num;
ods output lsmeancl = lsmn&num;
ods output lsmeandiffcl = est&num;
ods output estimates = pval&num;

%do l = 1 %to &blevels;

       data lsm&l;
              set &indata;
              &&wher&l
       run ;

       PROC GLM data=lsm&l;
              model &depvar =&clsvar;
              class &clsvar;
              model &depvar=&clsvar;
              means &clsvar;
              lsmeans &clsvar / &pdiff &stderr &alpha=&p &CL;

              /* CONTRAST/ESTIMATE statement linear vector of coefficients */
              estimate &&estmt&l
       run;
%end;

%mend mlsm;
```

CONCLUSION

The statistical analysis method for PK data is an example of a fairly standard analysis. Using a macro process to build SAS PROC GLM procedure statements can make the GLM robust to vary as the incoming data source varies. It also creates reusable code that results in cost and time savings in production and validation during iterative runs or when reusing code across studies or projects. The key is using the same set of procedures via MACRO parameter input over and over again without modification to ensure proper, accurate, validated analysis of the data.

REFERENCES

Chen, Huei-Ling and Yang, Aiming, "Common Pitfalls in SAS Statistical Analysis Macros in a Mass Production Environment." Proceedings of the 2007 North East SAS Users Group Conference, November 2007.

Lewis, Taylor, "PROC LOGISTIC: The Logistics Behind Interpreting Categorical Variable Effects." Proceedings of the 2007 North East SAS Users Group Conference, November 2007.

SAS Institute Inc. 2004. **SAS OnlineDoc® 9.1.2**. Cary, NC: SAS Institute Inc. (Advanced General Linear Models GLM procedure, Specification of Effects)

Su, Jing and Lin, Wei (Lisa), "Using Macro and ODS to Overcome Limitations of SAS Procedures." Proceedings of the 2007 North East SAS Users Group Conference, November 2007.

CONTACT INFORMATION

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

Frank Tedesco
Senior Statistical Programmer
United Biosource Corporation
920 Harvest Drive, Suite 200
Blue Bell, PA 19422
Tel: +1 (267) 470-1683
Fax: +1 (215) 591-2890
frank.tedesco@unitedbiosource.com