Using Macros to Build a Robust, Data Driven PROC GLM
Ginger Lewis, Frank Tedesco, United Biosource Corporation

Abstract:
A programmer can face many challenges when developing code to perform least square means tests with incomplete clinical trial data. They must review the data to determine class, independent, dependent and by variables to manually create the linear combination of coefficients for a single effect in the model. Each time data changes, the assumptions about variables and the L vector must be reviewed, potentially resulting in a need to modify the PROC GLM statements.

This paper presents a macro that provides a dynamic, data driven solution for constructing the L vector of coefficients for PROC GLM. The macro will use the parameters passed into the macro to determine the number of levels of each by variable and construct the CLASS, MODEL, LSMEANS, ESTIMATE and CONTRAST statements based on available data. It will evaluate the types of data available and build formats to accommodate numeric or character data. The macro will also create a summary of the statistical output of contrasts, means, confidence intervals and p-values.

This paper will also demonstrate that a robust macro program can provide program stability when data changes and create reusable code that will provide cost savings when applied across multiple clinical trials.

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 PROC GLM 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 evaluable requires a specific set of programmatic data handling rules and that patient evaluable 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.
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 (i.e., 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;
  %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 &byvar &clsvar;
      run;

      %let bcsortid=%sysfunc(open(work.sortcls,i));  /*** open the sorted data set ***/
      %let bclevels=%sysfunc(attrn(&bcsortid,nobs));   /*** determine the 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 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;  

Start GLM subsetting WHERE and start building 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 */  
    %end;  
    %let rc=%sysfunc(close(&csortid));  /* close the sorted data set */  
  %end;  

Now that you have the unique value of the BY variable, use a WHERE clause in the GLM to data &indata:  

%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");  

    %end;  
  %end;  

%end;
%put first part of where clause &whr;

%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 &BYVALN ***/ 
  %let whr=%str(where &by=&byvaln);
  %let whr_stg=%str(where &by=&byvaln);
  %put first part of where clause &whr;
  %end;
/* end of if then b=1 */

%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;
  %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; /*end of if then b>1*/
%end; /*end of b loop*/
%let wher&l=%str(&whr_stg;);
%put FINAL where clauses for each by variable combination wher&l=&&wher&l;
%end; /*end of l loop*/

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

Only keep BY variable-cohort-analyte-form combinations that satisfy analysis – linear combination of coefficients in model:

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

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

%let csortid=%sysfunc(open(work.sortcls&l,i));  /*** open the sorted data set ***/
%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)) = N %then
  %do;
  %let clsvln=%sysfunc(getvarn(&csortid,&clsvnm));
  %if &c=1 %then
    %do;
    %let constrn = &clsvln;
    %end;
  %else
    %do;
    %let constrn = %str(&constrn &clsvln);
    %end;
  %if &clsvln = &stdval %then %let location = &c;
  %end;
%end;
%if %sysfunc(vartype(&csortid,&clsvnm))= C %then
  %do;
  %let clsvlc=%sysfunc(getvarc(&csortid,&clsvnm));
  %if &c=1 %then
    %do;
    %let constrlc = &clsvlc;
    %end;
  %else
    %do;
    %let constrlc = %str(&constrlc &clsvlc);
    %end;
  %if &clsvlc = &stdval %then %let location = &c;
  %end;
%end;
%let fclsvnm = %sysfunc(varnum(&csortid,fmt&clsvar));
%if %sysfunc(vartype(&csortid,&fclsvnm))= C %then
  %do;
  %let fclsvlc=%sysfunc(getvarc(&csortid,&fclsvnm));
  %if &c=1 %then
    %do;
    %let fconstrlc = &fclsvlc;
    %end;
  %else
    %do;
    %let fconstrlc = %str(&fconstrlc &fclsvlc);
    %end;
  %if &fclsvlc = &stdval %then %let location = &c;
  %end;
%end;
%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;
%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 2nd bylevel loop*/
%end; /*end of byvar loop */

Use ODS statements to format and store statistics data of each run of the model. Use base SAS to store the string of L vector of coefficients in macro variables associated with the specific iteration of the PROC GLM and store in the form of macro variables.

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;

      ** add do loop for multiple class levels ****;
      estimate &&estmt&l;
      run;
   %end;
%end
mend mlsm;

Apply the selected option (macro variable) in the final run. Output the final report including all analyses from each iteration of GLM using ODS.
Summary:

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 reduces 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:


Key Words: clinical trial, cohort, analyte, PK, GLM, LSMEANS, ESTIMATE, CONTRAST

Contact Information:

The authors welcome your comments and questions. Please contact us at:

Ginger Lewis  
Sr. Manager, Clinical Programming  
ginger.lewis@unitedbiosource.com

Frank Tedesco  
Senior Clinical Programmer/Analyst  
frank.tedesco@unitedbiosource.com

United Biosource Corporation  
17 Blacksmith Rd.  
Newtown, PA 18940

Disclaimer:

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.