PharmaSUG China 2019 - Paper DM-048

Creating a Function for Lab Toxicity Grading using PROC FCMP

Richard Read Allen, Peak Statistical Services

ABSTRACT

This presentation will show a method for grading the toxicity of laboratory data using a user-defined function written using PROC FCMP. The procedure builds a function using DATA step code that can be stored and reused for multiple studies.

INTRODUCTION

The purpose of this paper is to introduce the reader to the FCMP procedure in SAS® for creating user-defined functions. It can be used to do repetitive tasks in the processing of clinical lab data. You will learn how to grade labs using a toxicity scale such as the CTCAE (Common Terminology Criteria for Adverse Events) or DMID (Division of Microbiology and Infectious Diseases) Adult Toxicity Table. Intermediate programming skills and a basic understanding of lab data in a clinical trial are beneficial.

BASICS OF PROC FCMP

The FCMP procedure (SAS Function Compiler) is part of Base SAS software beginning with version 9.2 and was originally developed as a programming language for several SAS/STAT, SAS/ETS, and SAS/OR procedures.

Using DATA step syntax PROC FCMP enables you to create, test, and store user-defined

- SAS functions
- CALL routines
- Subroutines

These are stored in a special type of data set and can be called in a DATA step and many SAS procedures. You can use most of the SAS programming statements and SAS functions that you can use in a DATA step to define FCMP functions and subroutines. The final code for your functions will be much easier to understand and maintain than the typical macro code usually used for such tasks.

In order to use functions defined by PROC FCMP, we need to store them in a special dataset (function library) with a four-level name as follows:

LIBREF . DATASET . PACKAGE . FUNCTION

where

LIBREF = library where dataset containing functions will be stored
DATASET = dataset within the library defined above containing the package of functions
PACKAGE = name of package containing functions
FUNCTION = name of function performing a specific task

It's possible to have multiple packages within a dataset and multiple functions within a package. For example, one could have a package called "conversions" and within this have a function for converting temperature from Fahrenheit to/from Celsius (LIBREF.DATASET.conversions.ConvertTemp) and another function for converting lab values to SI units (LIBREF.DATASET.conversions.ConvertLabs).

For SAS to find these functions we need to add the dataset to the search path using the CMPLIB system

option:

```
options cmplib = (LIBREF . DATASET);
```

This option is used when creating the functions to tell SAS where to store them and in the code when calling the functions to tell SAS where to find them. It's the same syntax in both situations.

When creating the functions in the PROC FCMP statement we then use the OUTLIB option to point to this dataset and the package within the dataset where we wish to store the functions.

```
PROC FCMP outlib = LIBREF . DATASET . PACKAGE;
```

To create a basic function in PROC FCMP, the following syntax and structure are used:

FUNCTION *function-name(argument-1, ..., argument-n)*;

... programming-statements ...

RETURN (*expression*);

ENDSUB;

In the programming statements above, most of the code you normally use in normal DATA step coding can be used to create your function. There are some exceptions which are explained in the PROC FCMP documentation. None of these affect what we plan to use in these simple examples. There are many other options and more detail available in PROC FCMP, but we will be concentrating on functions of this simple type in the examples that follow.

SAMPLE DATASET

We will use the sample dataset below to illustrate the concepts of creating a function to derive toxicity grades for some lab parameters.

usubjid	lbdtc	lbtestcd	lbstresu	lbstresn	lbstnrlo	lbstnrhi
1	03JAN2017	ALB	g/L	32	34	48
2	18AUG2017	ALB	g/L	44	34	48
1	03JAN2017	BILI	umol/L	11	0	25
2	18AUG2017	BILI	umol/L	32	0	25
1	03JAN2017	CREAT	umol/L	83	50	90
2	18AUG2017	CREAT	umol/L	110	50	90
1	03JAN2017	PLAT	x10E9/L	73	150	450
2	18AUG2017	PLAT	x10E9/L	329	150	450
1	03JAN2017	WBC	x10E9/L	6.6	3.5	11
2	18AUG2017	WBC	x10E9/L	2.7	3.5	11
1	28JUN2017	GLUC	mmol/L	2.9	3.9	7.7
2	26AUG2017	GLUC	mmol/L	9.3	3.9	7.7

TOXICITY GRADING OF LABS USING A FUNCTION

A very nice use of functions in clinical lab programming is to create a routine to derive the toxicity scores based on a standard toxicity grading system such as the CTCAE (Common Terminology Criteria for Adverse Events). It is easiest to apply this function after the conversions are done so that we only need to have grading equations for one type of units – SI units – for each lab parameter (LBTESTCD). It is also helpful to have identified the baseline observations for each parameter since some of the toxicity rules depend on the value of the baseline observation.

BASIC EQUATIONS FOR TOX GRADES

Below are the CTCAE toxicity grading scales for some of the parameters in our example:

Lab Parameter	Grade 1	Grade 2	Grade 3	Grade 4
Albumin (g/L)	<LLN - 30	<30 - 20	<20	Not applicable
Bilirubin (umol/L)	>ULN - 1.5 x ULN	>1.5 - 3.0 x ULN	>3.0 - 10.0 x ULN	>10.0 x ULN
Platelets (10^9/L)	<LLN -75.0	<75.0- 50.0	<50.0- 25.0	<25.0
White Cells Count (10^9/L)	<LLN-3	<3-2	<2-1	<1

Table 5. CTCAE Toxicity grading for selected lab parameters

When creating functions for grading toxicity, I prefer to use arithmetic expressions rather than a bunch of nested if/then/else logic. The programming statement to grade Albumin (ALB) would look like this:

```
if lbcd='ALB' then grade=3*( . <  value <  20)
                       +2*(20 <= value <  30)
                       +1*(30 <= value < uln);
```

This single statement returns the same result as the following 4 nested if/then/else statements

```
if lbcd='ALB' then do;
  if . <value <20 then grade=3;
  else if 20<=value<30 then grade=2;
  else if 30<=value<uln then grade=1;
  else grade=0;
end;
```

When designing the programming statements using arithmetic statements as above, one must be careful of missing limits in your data. The following code would be the full function to grade the toxicity levels for the four parameters above. Note that at the beginning flags NML and NMU are defined based on the existence of the limits in the data set. We used the NMU flag in the programming statement for BILI, otherwise if ULN is missing, all values would be graded 4.

```
proc fcmp outlib=funclib.functions.conversions;
  function GradeTox_CTCAE(lbcd $, lln, uln, value);
    nml=(missing(lln)=0);  *<== Non-missing normal range lower limit;
    nmu=(missing(uln)=0);  *<== Non-missing normal range upper limit;
   /*-------------------------------------------------------------*/
   /* Tox Grades
   /*-------------------------------------------------------------*/
   /*HEMATOLOGY */
     if lbcd='WBC' then grade=4*(       value < 1.0)
                           +3*(1.0 <= value < 2.0)
                           +2*(2.0 <= value < 3.0)
                           +1*(3.0 <= value < lln);
     else if lbcd='PLAT' then grade=4*( . <  value <  25)
                                +3*(25 <= value <  50)
                                +2*(50 <= value <  75)
                                +1*(75 <= value < lln);

  /*CHEMISTRY */
     else if lbcd='ALB' then grade=3*( . <  value <  20)
                               +2*(20 <= value <  30)
                               +1*(30 <= value < lln);

     else if lbcd='BILI' then grade=4*(nmu &            value >  10.0*uln)
                                +3*(nmu & 3.0*uln <  value <= 10.0*uln)
                                +2*(nmu & 1.5*uln <  value <=  3.0*uln)
                                +1*(nmu &     uln <  value <=  1.5*uln);

   return (grade);
  endsub;
run;
```

If we apply this function to our example data we would do it as in the following data step. We just call the function and supply the arguments in the order defined in the function above: (lbcd $, lln, uln, value). In the sample dataset this lbcd=lbtestcd, lln=lbstnrlo, uln=lbstnrhi and value=lbstresn.

```
data Example;
  set Sample;
*-- Call custom GradeTox_CTCAE function derive toxicity grades;
  lbtoxgrn=GradeTox_CTCAE(lbtestcd,lbstnrlo,lbstnrhi,lbstresn);
run;
```

After running this function our Example dataset will look like this:

usubjid	lbdtc	lbtestcd	lbstresu	lbstresn	lbstnrlo	lbstnrhi	lbtoxgrn
1	03JAN2017	ALB	g/L	32	34	48	1
2	18AUG2017	ALB	g/L	44	34	48	0
1	03JAN2017	BILI	umol/L	11	0	25	0
2	18AUG2017	BILI	umol/L	32	0	25	1
1	03JAN2017	CREAT	umol/L	83	50	90	
2	18AUG2017	CREAT	umol/L	110	50	90	
1	03JAN2017	PLAT	x10E9/L	73	150	450	2
2	18AUG2017	PLAT	x10E9/L	329	150	450	0
1	03JAN2017	WBC	x10E9/L	6.6	3.5	11	0
2	18AUG2017	WBC	x10E9/L	2.7	3.5	11	2
1	28JUN2017	GLUC	mmol/L	2.9	3.9	7.7	
2	26AUG2017	GLUC	mmol/L	9.3	3.9	7.7	

MORE COMPLICATED GRADING EQUATIONS

Other toxicity grading equations are a bit more challenging to program. In the example above, we have limited the equations to some of the simpler ones. However, there are more complicated grading rules some of which include comparisons to baseline values or other 'special' rules. The rule for CREAT is

Creatinine (umol/L)	>1 - 1.5 x baseline; >ULN -1.5 x ULN	>1.5 - 3.0 x baseline; >1.5 -3.0 x ULN	>3.0 baseline; >3.0 - 6.0 xULN	>6.0 x ULN

The programming statement for CREAT would require an extra argument be added to the function for the baseline value (BSLN). This statement would look something like this:

```
nmb=(missing(bsln)=0);  *<== Non-missing baseline value;

if lbcd='CREAT' then  grade=max(4*(   (nmu & value>6.0*uln)
                                   or (nmb & value>6.0*bsln)
                                   )
                               ,3*(   (nmu & 3.0*uln<value<= 6.0*uln)
                                   or (nmb & 3.0*bsln<value<=6.0*bsln)
                                   )
                               ,2*(   (nmu & 1.5*uln<value<= 3.0*uln)
                                   or (nmb & 1.5*bsln<value<=3.0*bsln)
                                  )
                               ,1*(   (nmu &     uln<value<=1.5*uln)
                                   or (nmb &     bsln<value<= 1.5*bsln)
                                  )
                               );
```

The reason to use the maximum rather than the sum is that it is possible for the value to satisfy one condition compared baseline and a different condition based on the value and ULN. Note also the NMB is a check for non-missing baseline value similar to the check for NML and NMU is the original function.

Other lab parameters also have separate rules for values out of lower range (HYPO) and out of upper range (HYPER). One such parameter is Glucose

Glucose (mmol/L)	HYPO	<LLN - 3.0	<3.0 - 2.2	<2.2 - 1.7	<1.7
Glucose (mmol/L)	HYPER	>ULN - 8.9	>8.9 - 13.9	>13.9 -27.8	>27.8

Programming statements to grade GLUC would look like this and does both.

```
                            /**  #Hypoglycemia            #Hyperglycemia **/
   if lbcd='GLUC' then grade=4*(( . < value<1.7) or (     27.8<value      ))
                            +3*((1.7<=value<2.2) or (     13.9<value<=27.8))
                            +2*((2.2<=value<3  ) or (      8.9<value<=13.9))
                            +1*((  3<=value<lln) or (nmu & uln<value<=8.9));
```

These can be flagged as HYPO or HYPER depending whether LBFLAG is L or H, respectively.

If we incorporate these into the above function, we'd have to add another argument for the baseline value as follows:

```
  function GradeTox_CTCAE(lbcd $, lln, uln, bsln, value);
```

Then if we run the function as in this example, the output for CREAT and GLUC will be as below:

```
Example2;
  set Sample;
  if lbtestcd='CREAT' then base=85;
*-- Call custom GradeTox_CTCAE function derive toxicity grades;
  lbtoxgrn=GradeTox_CTCAE(lbtestcd,lbstnrlo,lbstnrhi,base,lbstresn);
run;
```

usubjid	lbdtc	lbtestcd	lbstresu	lbstresn	lbstnrlo	lbstnrhi	base	lbtoxgrn
1	03JAN2017	CREAT	umol/L	83	50	90	85	**0**
2	18AUG2017	CREAT	umol/L	110	50	90	85	**1**
1	28JUN2017	GLUC	mmol/L	3.1	3.9	7.7		**1**
2	26AUG2017	GLUC	mmol/L	9.3	3.9	7.7		**2**

EVEN MORE COMPLICATED GRADING EQUATIONS

Below are a few rules from the DMID Adult Toxicity Table (November 21, 2007) that involve 'special' rules in bold. When developing functions for these types of grading one need to work these special rules into the equations.

Lab Parameter	Grade 0	Grade 1	Grade 2	Grade 3	Grade 4
Hypoglycemia (mg/dL)	≥65	55-64	40-54	30-39	<30
Hyperglycemia (mg/dL) (**nonfasting and no prior diabetes**)	<116	116-160	161-250	251-500	>500
Hyperbilirubinemia (**when accompanied by any increase in other liver function test**)	<1.1xULN	1.1 - <1.25xULN	1.25 - <1.5xULN	1.5 – 1.75xULN	> 1.75xULN
Hyperbilirubinemia (**when other liver function are in the normal range**)	<1.1xULN	1.1 - <1.5xULN	1.5 - <2.0xULN	2.0 – 3.0xULN	> 3.0xULN

The function as defined below will do the grading of GLUC and BILI using these rules. Another argument for the special rules has been added to the function

```
proc fcmp outlib=funclib.functions.conversions;
  function GradeTox_DMID(lbcd $, nrlo, nrhi, bsln, Special, value);
    nmb=(missing(bsln)=0); *<== Non-missing baseline;
    nml=(missing(nrlo)=0); *<== Non-missing normal range low;
    nmh=(missing(nrhi)=0); *<== Non-missing normal range high;
   /*-------------------------------------------------------------------*/
   /* Tox Grades
   /*-------------------------------------------------------------------*/
                       /**   #Hypoglycemia        #Hyperglycemia /
                                                  non-fasting and no
                                                  prior diabetes       **/

  if lbcd='GLUC' then grade=4*((. < value<30) | (     value> 500 & Special))
                          +3*((30<=value<40) | (250< value<=500 & Special))
                          +2*((40<=value<55) | (160< value<=250 & Special))
                          +1*((55<=value<65) | (116<=value<=160 & Special))
                          +0*((65<=value   ) | (     value< 116 & Special));


   /**   #Hyperbilirubinemia / with increase in other liver functions  **/
  else if lbcd='BILI' & Special=1 then
                      grade=4*(nmh & 1.75*nrhi < value           )
                          +3*(nmh & 1.50*nrhi < value<=1.75*nrhi)
                          +2*(nmh & 1.25*nrhi < value<=1.50*nrhi)
                          +1*(nmh & 1.10*nrhi <=value<=1.25*nrhi)
                          +0*(nmh &             value< 1.10*nrhi);
```

```
   /**   #Hyperbilirubinemia / with no increase in other liver functions  **/
  else if lbcd='BILI' & Special=0 then
                      grade=4*(nmh & 3.0*nrhi < value          )
                           +3*(nmh & 2.0*nrhi < value<=3.0*nrhi)
                           +2*(nmh & 1.5*nrhi < value<=2.0*nrhi)
                           +1*(nmh & 1.1*nrhi <=value<=1.5*nrhi)
                           +0*(nmh &            value< 1.1*nrhi);

   return (grade);
  endsub;
run;
```

Note that the units for GLUC are mg/dL, so these must be converted from mmol/L by multiplying by 18 before applying the function. Also one must derive the requirements for the special rules and include a 0/1 variable indicating whether or not the special rules are met in the input dataset. This is the variable RQMNT in the example below..

```
data Example2;
  set Sample;
*-- Call custom GradeTox_DMID function derive toxicity grades;
  lbtoxgrn=GradeTox_DMID(lbtestcd,lbstnrlo,lbstnrhi,base,rqmnt,lbstresn);
run;
```

When we apply the function to the sample data below, the results can be seen in the last column.

usubjid	lbdtc	lbtestcd	lbstresu	lbstresn	lbstnrlo	Lbstnrhi	rqmnt	lbtoxgrn
1	03JAN2017	BILI	umol/L	11	0	25	**0**	**0**
2	18AUG2017	BILI	umol/L	32	0	25	**1**	**2**
3	29AUG2017	BILI	umol/L	32	0	25	**0**	**1**
1	28JUN2017	GLUC	mg/dL	56	70	139	**1**	**1**
2	26AUG2017	GLUC	mg/dL	167	70	139	**0**	**0**
3	08SEP2017	GLUC	mg/dL	167	70	139	**1**	**2**

A third subject has been added to this example with the same LBSTRESN as the second subject but one satisfies the requirement and the other does not so that the effect of the special rule can be seen.

CONCLUSION

The features of PROC FCMP enable clinical programmers to more easily read, write and maintain complex code with independent and reusable subroutines. It is especially useful in processing lab data.

Functions can be expanded and modified easily. They use regular DATA step code so they are easier to understand and maintain than the equivalent MACRO code.

You can reuse the PROC FCMP routines in any DATA step or SAS procedure that has access to their storage location.

REFERENCES

Using PROC FCMP to the Fullest: Getting Started and Doing More, Arthur L. Carpenter, California Occidental Consultants, Anchorage, AK – HOW presented at WUSS 2013.

Base SAS 9.4 Procedures Guide, Second Edition, SAS Institute, Cary, NC

CTCAE vs Laboratory Parameters, PhUSE Wiki, http://www.phusewiki.org/wiki/index.php?title=CTCAE_criteria_vs_laboratory_parameters

ACKNOWLEDGMENTS

Many thanks to Art Carpenter whose HOW at WUSS in 2013 started my thinking of uses of PROC FCMP in my daily work.

RECOMMENDED READING

- *Hashing in PROC FCMP to Enhance Your Productivity, Andrew Henrick, Donald Erdman and Stacey Christian, SAS Institute, Cary, NC – Presented at WUSS 2014*

CONTACT INFORMATION

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

Richard Read Allen
Peak Statistical Services
303-670-5386
rrallen@peakstat.com
www.peakstat.com