Statistical Development Using SAS/TOOLKIT® Software

Scott D Grimshaw, Department of Statistics, Brigham Young University, Provo, UT
Gilbert W. Fellingham, Department of Statistics, Brigham Young University, Provo, UT

Abstract

SAS/TOOLKIT provides the ability to include user-written procedures, functions, CALL routines, formats and informats into the SAS® System. This paper presents a user-written SAS/IML® function written in C on the UNIX platform. Sample C code is presented for a user-written SAS/IML function to compute the number of runs in a vector of zeros and ones. SAS/TOOLKIT is used to create a SAS executable module from the compiled C code. This example is demonstrated in a small SAS/IML program.

Introduction

Many statisticians use SAS/IML to develop statistical methodologies. SAS/IML provides the data-handing tools and a rich collection of functions and CALL routines which can be used to perform many different statistical calculations. At SUGI 22 in the Statistics, Data Analysis, and Modeling section, SAS/IML was used to perform real-time process monitoring, Gibbs sampling in regression, higher-order asymptotic *p*-values in mixed models, logistic regression in complex surveys, balanced replicated subsampling for assessing error variances from complex survey samples, E-M estimation in a bivariate random effects model, Kruskal's relative importance analysis, analysis of combined tables of categorical data, maximum likelihood estimation in the TOBIT model, and bootstrap confidence intervals for the kappa coefficient.

SAS/TOOLKIT, an optional applications development product in the SAS system, enables a user to write routines in C, FORTRAN, PL/I, or assembler that can be executed within SAS programs. While many SAS/IML functions exist and SAS/IML can be used to write programs for statistical development, there are several advantages to a user-written function: (1) functions are written and supported in only one language instead of a separate language for each statistical software package, (2) developing statistical algorithms in SAS/IML is possible but the execution is sometimes slow, (3) some operations are easy to program in another language but difficult in SAS/IML.

In this paper, a C routine is presented to compute the number of runs in a vector of zeros and ones. The necessary features and functions of this code are documented and described. SAS/TOOLKIT is used to compile and link the C code to form an executable module recognized by the SAS supervisor. A sample SAS/IML program is provided to demonstrate the user-written function call in a statistical development environment.

Structure of User-Written IML Functions

User-written SAS/IML functions are organized in executable modules, with each executable module containing the user-defined functions and CALL routines. The SAS supervisor recognizes the entire module by the name UWUxchars, where the UWU indicates the executable module contains functions or CALL routines and xchars are the first five characters common to all functions or CALL routine names. For example, all the user-written SAS/IML functions could be combined into the executable module UWUMYFNC.

Within an executable module, up to 64 functions or CALL routines can be defined. Each function or CALL routine can be up to eight characters, but the first five must match the xchars component of the executable module name. For example, within the executable module UWUMYFNC, each user-written SAS/IML function is of the form MYFNCychars. The following are acceptable names in UWUMYFNC: MYFNC1, MYFNCXY, MYFNCRUN. The following would be unacceptable names in UWUMYFNC: 1MYFNC, MYFNCXYZ123, RUNSTEST. Clearly this makes the effort to organize functions into similar syntax and functionality worthwhile. On the other hand, an executable module could contain only one SAS/IML function so a name might be very specific.

Template for C Code

Many C routines are provided by SAS/TOOLKIT for the creation of executable modules and data operations within a SAS/IML function. In the example below, the routines FNCDFS and FNCDFI define the number, names, inputs and outputs of the user-written SAS/IML functions and call routines in the executable module. Within the definition of the first (and only function in this example) user-written SAS/IML function called MYFNCRUN, the routines SAS_IMWRES and SAS_IMWARG allocate memory to the returned value and bring the input matrix into the C routine.

```
/*-------------------------------------*/
/* NAME:         MYFNC                 */
/* PRODUCT:      SAS/TOOLKIT           */
/* LANGUAGE:     C                     */
/* MODULE:       MYFNC                 */
/* TYPE:         IML FUNCTIONS         */
/* IML FUNCTIONS IN THIS MODULE:       */
/*    MYFNCRUN                         */
/*    PURPOSE:   IML function to       */
/*               compute the number    */
/*               of runs in a vector   */
/*               of zeros and ones     */
/*                                     */
/*    USAGE:     r = myfncrun(x);      */
/*                                     */
/*               where                 */
/*                                     */
/*               x = input vector of   */
/*                   0's and 1's       */
/*               r = output tally of   */
/*                   the number of     */
/*                   runs in x         */
/*                                     */
/*-------------------------------------*/

#include "uwproc.h"

ptr IFFMAI(request)
int *request;
{
if (*request == 1) {
   UWPRCC(0);
```

```
/* use FNCDFS to declare the number   */
/* functions defined in this module   */
   FNCDFS(1);
/*        ^                           */
/*        |                           */
/*        number of functions defined */
/*        in this module              */

/* use FNCDFI to declare the          */
/* characteristics of the SAS/IML     */
/* function                           */
   FNCDFI(1,"MYFNCRUN",1,1,1,1);
/*        ^     ^      ^ ^ ^ ^        */
/*        |     |      | | | |        */
/*        |     |      | | | number of*/
/*        |     |      | | | returned */
/*        |     |      | | | values   */
/*        |     |      | | declare a  */
/*        |     |      | | CALL       */
/*        |     |      | | routine(0) */
/*        |     |      | | or a       */
/*        |     |      | | FUNCTION(1)*/
/*        |     |      | maximum      */
/*        |     |      | number of    */
/*        |     |      | arguments in */
/*        |     |      | IML function */
/*        |     |      minimum number */
/*        |     |      of arguments   */
/*        |     |      in IML function*/
/*        |     name of the           */
/*        |     IML function          */
/*        number of the function in   */
/*        the module                  */

   return(FNCDFE());
   }
else if (*request == 2) {
   return(FNCFNE());
   }
}

/* each function in the module is     */
/* recognized internally as RTNi      */
/*                                    */
/* RTN1 identifies the C function     */
/* which is executed during a call    */
/* in IML to MYFNCRUN to count the    */
/* number of runs in a vector         */

int RTN1(arg, to_n)

   ptr *arg;
   int *to_n;

{
   long    i;
   int     row, col, size;
   dblptr  r1, x;
   double  runs;

/* use SAS_IMWRES to allocate memory  */
/* for the return value r1            */
/*                                    */
/* in this case, the returned value   */
/* is a scalar (1x1 matrix)           */
   r1 = SAS_IMWRES(1,1,1);
/*                 ^ ^ ^              */
/*                 | | |              */
/*                 | | number of      */
/*                 | | columns of r1  */
/*                 | number of        */
/*                 | rows of r1       */
/*                 number of the      */
/*                 result             */

   if (!r1) return (IML_MEMORY);

/* use SAS_IMWARG to bring the        */
/* vector x in the IML function call  */
/* into the C subroutine              */
   SAS_IMWARG
   (arg[0],&row,&col,&size,&x);
/*  ^      ^    ^     ^    ^          */
/*  |      |    |     |    |          */
/*  |      |    |     |    matrix     */
/*  |      |    |     |    address of */
/*  |      |    |     |    x returned */
/*  |      |    |     element size    */
/*  |      |    |     returned        */
/*  |      |    number of columns     */
/*  |      |    of x returned         */
/*  |      number of rows             */
/*  |      of x returned              */
/*  symbol table address              */

/* algorithm to count the number of   */
/* runs in x                          */
   runs = 1;
   for (i=1; i<row; i++){
      if ( *(x+i) != *(x+i-1) )
         runs += 1;
   }

/* declare the value of runs to be    */
/* the return value r1                */
   *r1 = runs;

   return(IML_OK);

}
/* Other functions in the module      */
/* would follow next                  */
/* (RTN2, RTN3, ..., RTN64)           */
void IFFEXT() {}
```

Compiling and Linking into an Executable Module in UNIX

On the UNIX platform, the make command greatly simplifies the required tasks of compiling, forming a program constants object, creating a SAS grammar object, and linking all these components to form the executable module.

The example make file given below assumes a development directory myfnc with subdirectories: myfnc/src containing the C source code files, myfnc/maclib containing the SAS/TOOLKIT C libraries, myfnc/cntl containing the UNIX make files, myfnc/obj containing the compiled C objects and SAS compiled grammars, and myfnc/load containing the user-written executable modules.

In the example below, the declarations of IFFOBJ, MODNAME, PGMCON are used to link the compiled C object, program constants object, and SAS grammar object. IFFOBJ identifies the name of the compiled C object. MODNAME names the executable module in the form UWUxchars previously discussed. PGMCON names the SAS grammar object to be created by SAS/TOOLKIT.

In the local make definitions, the names of the C source code and compiled objects must be declared. This allows the make file to move the necessary files around in the development directories.

```
#######################################
# NAME:    myfnc.make                 #
# PRODUCT: SAS/TOOLKIT                #
# PURPOSE: make file for creating the #
#          user-written executable    #
#          model containing           #
#          user-written IML functions #
#######################################
include toolkit.make
# local macros                        #
# name of the compiled C object       #
IFFOBJ          = $(OBJ)myfnc.o

# name of the executable module in    #
# the form UWUxchars                  #
MODNAME         = uwumyfnc

# name of the SAS                     #
# program constants object            #
PGMCON          = mcbuwumy

UWH             = $(UWPROCH)
WHICH           = FUNCTION
LANGUAGE        = c
PRCINT          = $(PRCINTCV)
include where.make

# local MAKE definitions              #

# names of C code source              #
# and object files                    #
ALL : $(LOAD)$(MODNAME)
$(OBJ)myfnc.o : $(SRC)myfnc.c $(UWH); \
      $(UWC) $(SRC)myfnc.c $(UWCOPTS);\
                  mv myfnc.o $(OBJ)

include pgmcon.make
include uwlnkiff.make
#######################################
```

The toolkit.make file which is included in this sample make file includes tlkthost.make which may need to be edited to match a particular site installation. For example, at our site the INCLLIB, TLKTDIR, TLKTSAMP declarations were changed to point to the location of the TOOLKIT product in sas612/saspgm/toolkit. The declaration of SASCMD was also changed to sas.

After executing the myfcn.make file using the UNIX command make -f myfcn.make the executable module uwumyfcn is created in the myfcn/load development subdirectory. Change this file mode to executable using the UNIX chmod command.

The executable module must be available in the search path of the SAS supervisor. For example, the SAS configuration file can be edited to include the myfnc/load subdirectory. Other alternatives include using the -path option to point to the directory containing the executable module or moving the executable module to one of the SAS executable directories.

The file uwumyfnc can be circulated to any SAS user on the same platform for execution. The users receiving the executable module do not need SAS/TOOLKIT.

User-Written Function in SAS/IML

In the following example, the user-written function myfncrun is called to compute the number of runs in the vector x.

```
/*   Sample SAS Program Demonstrating */
/*   a User-Written IML Function      */
proc iml;

/*   x vector of zeros and ones       */
x={1,1,0,0,1,1,1,0,1,0,1,0,1,1,1};

/*   call the user-defined function   */
/*   MYFNCRUN to compute the          */
/*   number of runs in the vector x   */
r=myfncrun(x);

print 'Number of Runs:' r;

/*   the correct number of runs is 9  */

endsas;
```

Conclusion

This paper has demonstrated the use of SAS/TOOLKIT to create and call user-written SAS/IML functions. This feature is valuable to statisticians developing methodologies because SAS/IML provides the data handling features and the development is focused on operations that may be slow or impossible using SAS/IML functions. The structure of user-written functions and CALL routines into executable modules has been described. A sample of C code for a user-written SAS/IML function to compute the number of runs in a vector of zeros and ones is provided with important features described. Under UNIX, the compiling and linking operations can be performed with the make command. A sample make file is given. Finally, the use of the user-written SAS/IML function is demonstrated in a small example.

References

SAS Institute Inc. (1991*), SAS/TOOLKIT Software: Usage and Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1992), *SAS Technical Report P-245, SAS/TOOLKIT Software: Changes and Enhancements, Releases 6.08 and 6.09*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1997), *Proceedings of the Twenty-Second Annual SAS users Group International Conference*, Cary, NC: SAS Institute Inc.

For further information about this paper, contact:

Scott D Grimshaw
Department of Statistics
Brigham Young University
Provo, UT 84602
(801) 378-6251
grimshaw@byu.edu

Gilbert W. Fellingham
Department of Statistics
Brigham Young University
Provo, UT 84602
(801) 378-2806
gwf@byu.edu