ABSTRACT

This paper introduces the concept of interfacing SAS® language code to external subroutine libraries such as the Engineering and Scientific Subroutine Library (ESSL) from IBM®. This interfacing is accomplished by developing user-written SAS/IML® functions with the SAS/TOOLKIT™ product.

INTRODUCTION

A commonly requested feature is the capability of referencing external subroutines from within SAS code. This could be within a DATA step function, an IML function, or a Screen Control Language (SCL) function. An example of an external subroutine would be one of any of the ESSL subroutines provided by IBM. These subroutines handle arguments that are scalars, vectors (that is, one-dimensional arrays), and matrices (that is, two-dimensional arrays).

INTERFACING PROBLEMS

There are several problems involved in interfacing external subroutines with the SAS language. First, most subroutines are expected to be linked into some other program, and called only from that main program. And since the SAS System invokes its functions by means of linked modules, it is necessary that the desired subroutine first be linked into such a module that will properly call it. Second, there is the problem of language environment. Many of the external subroutine libraries, ESSL included, are designed to be used with the FORTRAN language. It may be desirable, or even required, that the calling subroutines be written in languages other than the expected implementation language so that language interfacing becomes an issue. Third, one must consider whether the particular SAS environment (DATA step, IML step, SCL step) can handle the proper aggregate that the external subroutine expects. For example, consider an external subroutine that expects a vector or matrix argument. The DATA step does not pass such entities to its called functions; it passes only scalar values. Therefore, external subroutines that expect vectors and matrices are limited to those portions of the SAS System that handle such aggregates, for example, IML steps. If the subroutine only expects scalars, both IML and SCL steps can access a user-written DATA step function successfully.

Once you have decided what external subroutine to call, what language to use in implementation, and in what context (DATA step, IML step, SCL step) it will be called, you can use the SAS/TOOLKIT™ software to help develop the function. This paper demonstrates the use of the DGEF subroutine from IBM's ESSL library. The paper also uses C as the implementation language, using the SAS/C® compiler for processing the C code.

THE ESSL EXAMPLE

The ESSL DGEF subroutine performs general matrix factorization by Gaussian elimination. Its calling sequence in FORTRAN is

\[ \text{CALL DGEF(A, LDA, N, IPVVT)} \]

where \( A \) is the matrix to be factored, \( LDA \) is the size of the leading dimension (that is, the maximum number of rows), \( N \) is the actual number of rows, and \( IPVVT \) is a pivot vector whose values are computed by DGEF. The matrix \( A \) will be changed by DGEF to contain the new factored values. The matrix \( A \) and its result are expected to contain \( N \) rows and \( N \) columns. \( IPVVT \) will have \( N \) cells. \( A \) is expected to contain REAL*8 (double) values, while the IPVVT vector will contain INTEGER*4 (long) values upon return. Both \( LDA \) and \( N \) are expected to be INTEGER*4 values. The LDA value is the total row count that \( A \) could contain, while \( N \) is the row count that DGEF will actually use. LDA and \( N \) are usually the same, unless only a portion of \( A \) is to be factored.

SAS/IML software can handle scalars, vectors, and matrices quite naturally. A scalar is a 1 X 1 matrix, and a vector is a 1 X \( N \) matrix. However, SAS/IML
The software expects all numeric matrices to contain doubles, not integers. This means that the IML function you write must convert input arguments to and from integer values where necessary.

**ARRAY STORAGE**

Another consideration concerns the storage of two-dimensional matrices. Since SAS/IML software is implemented using the C language, the native representation of matrices for that language is used, which is row-major order. Row-major order indicates that the array is stored in memory one row at a time. Consider a two-dimensional matrix \( X \) with two rows and two columns. \( X[0][0] \) is 1, \( X[0][1] \) is 2, \( X[1][0] \) is 3, and \( X[1][1] \) is 4. The four values composing the entire matrix will be stored in memory in the order 1, 2, 3, 4. This is not the same storage method used by FORTRAN, which instead uses column-major order. Column-major order indicates that the array is stored in memory one column at a time. This means that the example matrix will be stored in memory as 1, 3, 2, 4. In short, you must have to transpose any matrix that is to be passed to or from the IML step to or from FORTRAN subroutines. No manipulation of vectors or scalars is necessary unless FORTRAN expects integers. Note that compressed format also differs between languages, but is not discussed in this paper.

Some subroutines in the ESSL library do enable you to pass matrices that will be treated in transposed fashion by the ESSL subroutine. If you are implementing interfaces to such routines, you might be able to skip the transposition of certain matrices. The ESSL documentation indicates which subroutines allow such matrices in transposed format by a 'T' in the calling sequence definitions.

**THE USER-WRITTEN IML FUNCTION**

With these restrictions in mind, you must now decide how the function will look to the user of SAS/IML software. You want the calling sequence for the IML function to look as much as possible like the calling sequence for DGEEF as described in the ESSL documentation. However, SAS/IML software requires that all the result matrices come first in the calling sequence. DGEEF has two result matrices: \( A \) and \( IPVT \). \( A \) is also an input matrix, but these must be separated for the IML function. LDA is not usually necessary, since you can determine the maximum number of rows from IML utility functions within the implemented function. Nonetheless, the LDA argument is left for consistency's sake. \( N \) will be provided in case the caller wants to factor only a portion of the matrix. Now, the new IML function will be called like this:

```plaintext
call dgef(R,IPVT,A,LDA,N);
```

Note that you are using the same name for the IML function as the ESSL function, which is permissible. LDA and \( N \) are also allowed to to have missing values, or to be omitted. In either case you'll get their values from the row count of \( A \).

It is desirable to allow the IML user to reference the DGEEF function the same way it is documented in the ESSL manual. To allow this, you can create a SAS macro to convert the user's calling sequence to the one expected by your IML function, then reset the \( A \) matrix after the call to your function:

```plaintext
%macro _dgef(a,lda,a,ipvt);
  call dgef(temps,ipvt,a,lda,temps);
  free temps;
%mend;
```

Now you can begin to develop the actual C code that compiles the user-written IML function. The code is explained a portion at a time in the subsequent text.
value 1 is passed when information is requested about all functions being defined in the source. The value 2 is passed to obtain the function pointers. Most of the work in communicating between the SAS System and the function is done by the SAS/TOOLKIT code that is linked into your module. However, you are responsible for providing the definitions for your functions. Above, FNCDFS was called to indicate how many functions will be defined. There is only one function in our example. Then, we call FNCDFI to define an IML function. You indicate that it is the first function being defined, and that its name is DGEF. It has a minimum and a maximum input argument count of 3. It has no return attribute, therefore it is a CALL routine. There are two result matrices. The calls to FNCDFE and FNCFNE indicate to SAS/TOOLKIT that all definitions are in place.

Now begins the actual function. The first function must be called RTN1, the second RTN2, and so on. All IML functions are implemented having two arguments to the first being an argument pointer, and the second being a special value not used by the IML function writer. The argument pointer points to an array of anchor pointers that are used in various IML service routines that the implementer will call, whose names begin with SAS.IMW.

Following is the list of variables that will be used in the program. They appear here to you better understand their use later in the example.

It is a good habit to use function prototypes whenever possible, to ensure that you are passing all arguments correctly. The SAS/C Compiler will warn you if your arguments do not match those in the prototype, either in the function call or function definition. ARG and XPOSE are two internal functions that we will define later in the example. DGEF is the ESSL function. Note that all of its arguments have pointer attributes. This is because FORTRAN is a call-by-address language, and expects all the arguments in its parameter list to be pointers to the actual arguments.
Now you can acquire the result matrices. Here, you must know the number of rows and columns, and you pass this information to the IML supervisor by means of the SAS.IMWRES service routine, indicating which result matrix is being defined, along with its number of rows and columns. The result matrices are numbered starting with 1, so result matrix 1 will be the R matrix that will contain the factored A. R will have the same number of rows and columns as A, and that number will be N. If the return pointer from SAS.IMWRES is NULL, this means that the IML supervisor could not acquire the memory for the result, so you must return the IML_MEMORY return code.

Next, acquire the result for the pivot vector. It is result number 2, and has 1 row (since it is a vector), and has N as the column count.

As discussed above, you must transpose A before sending it to DGEF. A rule within IML functions is that you do not alter the contents of an input matrix, so you must allocate temporary memory for a transposed copy of A.

Also, since DGEF will expect the memory for a vector of longs for JPVT, you must allocate that memory. It has N elements.

Because memory was allocated after results, the result matrices must be re-resolved. It is possible that the IML supervisor relocated them in the garbage collection process.

Likewise, the IML supervisor may have relocated the input matrices. Therefore, you need to call the service routine SAS.IMWARG to obtain their addresses. First, you obtain A's address, then call the XPOSE routine (defined below) to transpose A into the temporary memory allocated above. There is no need to reacquire LDA and N, since those values have already been computed earlier.

Now that you have the transposed matrix, an integer value of LDA and N, and the memory for an integer vector for JPVT, call the DGEF ESSL routine. Call it via the CALLFTN routine, which will be responsible for the proper interfacing to the FORTRAN framework. Pass the function name as the first argument, with all subsequent arguments composing the proper calling sequence for DGEF as described in the ESSL documentation.

The integer vector now must be converted to a double vector. Do this by stepping through each element and casting it.

All that is left is to free the memory allocated by SAS.IMWAL02 above. If this freeing is not done, there could be fragmentation that would affect the IML user during the remaining IML session.
If you got this far, you can return a zero to indicate success.

return(0);
}

Here is a description of the internal routines. ARG is simply a shorthand routine that calls a couple of IML service routines. Acquire the row and column count for the matrix, the size of the matrix element, and the address of the matrix. If that returned address is NULL, this means that either the argument was omitted, or it was an undefined matrix. Return the IML_NULLMATRIX return code so that the IML supervisor can print the proper message if necessary. (Note that in the case of LDA and N, omission is acceptable.) The size value will be a positive number if the matrix is a character matrix. It will be -8 if you have a proper matrix of doubles. Since the code will allow only double values, return the IML_NOTNUMERIC return code if you encounter an incorrect matrix attribute.

The IFFEXT routine is required to exist, even if it does not do anything. The IFFEXT routine is called by the step termination routines so that you can terminate any environment, free memory, or perform any other kind of necessary cleanup. In the example, you need to call CALLFTN one last time. If that routine is called with a NULL argument, this indicates that the FORTRAN framework is to be terminated. If CALLFTN was not called, there would be problems with subsequent invocation of this function in another IML step.

The user-written IML function is compiled and linked using the same method as user-written informats/formats/functions/CALL routines (IFFCs) with the SAS/TOOLKIT software. For brevity, compiling and linking of the module are not discussed.

**EXAMPLE OF USE WITH AN IML STEP**

Here is an example of the use of the completed DGEF IML call routine. The same data are used as described in the DGEF example in the ESSL documentation.

The previously discussed macro is included to allow for easier user interfacing. It is followed by the IML step that defines the A matrix, calls the DGEF routine, and prints out the result matrices.

```
1 %macro dgef(a,llda,n,ipvt);
2 call dgef(tempe,ipvt,a,lda,km);
3 km = tempa;
4 free tempa;
5 %end;
6 options nocenter macrogen;  
7 proc iml;
8 IML Ready
9 a = { 1 1 1 1 0 0 0 0 0 ,
  1 1 1 1 1 0 0 0 0 ,
  4 1 1 1 1 1 0 0 0 ,
  0 5 1 1 1 1 1 0 0 ,
```
CONCLUSION

You may have subroutines that you want to access in a way similar to this example. If the desired subroutine is in FORTRAN, use the CALLFTN routine to interface to that language framework properly, and convert any arguments (by transposition or numeric conversion as appropriate) before calling the routine. Remember that if your desired subroutine uses only scalar arguments, you can write a DATA step function (which can also be used in an IML step). However, if your subroutine needs vectors or matrices, you will need to write an IML function to properly pass arguments directly. For either DATA step functions or IML functions, use the SAS/TOOLKIT software to compile and link your modules.

SAS, SAS/IML, and SAS/C are registered trademarks and SAS/TOOLKIT is a trademark of SAS Institute Inc, Cary, NC, USA.

IBM is a registered trademark of International Business Machines Corporation.