Chapter 16
Further Notes

Chapter Table of Contents

Chapter 16
Further Notes

Memory and Workspace

You do not need to be concerned with the details of memory usage because memory allocation is done automatically. However, if you are interested, the following sections explain how it works.

There are two logical areas of memory, *symbolspace* and *workspace*. Symbolspace contains symbol table information and compiled statements. Workspace contains matrix data values. Workspace itself is divided into one or more extents.

At the start of a session, the symbolspace and the first extent of workspace are allocated automatically. More workspace is allocated as the need to store data values grows. The SYMSIZE= and WORKSIZE= options in the PROC IML statement give you control over the size of symbolspace and the size of each extent of workspace. If you do not specify these options, PROC IML uses host dependent defaults. For example, you can begin an IML session and set the SYMSIZE= and WORKSIZE= options with the statement

```
proc iml symsize=n1 worksize=n2;
```

where $n1$ and $n2$ are specified in kilobytes.

If the symbolspace memory becomes exhausted, more memory is automatically acquired. The symbolspace is stable memory and is not compressible like workspace. Symbolspace is recycled whenever possible for reuse as the same type of object. For example, temporary symbols may be deleted after they are used in evaluating an expression. The symbolspace formerly used by these temporaries is added to a list of free symbol-table nodes. When allocating temporary variables to evaluate another expression, IML looks for symbol-table nodes in this list first before consuming unused symbolspace.

Workspace is compressible memory. Workspace extents fill up as more matrices are defined by operations. Holes in extents appear as you free matrices or as IML frees temporary intermediate results. When an extent fills up, compression reclaims the holes that have appeared in the extent. If compression does not reclaim enough memory for the current allocation, IML allocates a new extent. This procedure results in the existence of a list of extents, each of which contains a mixture of active memory and holes of unused memory. There is always a current extent, the one in which the last allocation was made.

For a new allocation, the search for free space begins in the current extent and proceeds around the extent list until finding enough memory or returning to the current extent. If the search returns to the current extent, IML begins a second transversal of the extent list, compressing each extent until either finding sufficient memory or returning to the current extent. If the second search returns to the current extent, IML opens a new extent and makes it the current one.

If the SAS System cannot provide enough memory to open a new extent with the full extent size, IML repeatedly reduces its request by 2K. In this case, the successfully opened extent will be smaller than the standard size.

If a single allocation is larger than the standard extent size, IML requests an allocation large enough to hold the matrix.

The WORKSIZE= and SYMSIZE= options offer tools for tuning memory usage. For data intensive applications involving a few large matrices, use a high WORKSIZE= value and a low SYMSIZE= value. For symbol intensive applications involving many matrices, perhaps through the use of many IML modules, use a high SYMSIZE= value.

You can use the SHOW SPACE command to display the current status of IML memory usage. This command also lists the total number of compressions done on all extents.

Setting the DETAILS option in the RESET command prints messages in the output file when IML compresses an extent, opens a new extent, allocates a large object, or acquires more symbolspace. These messages can be useful because these actions normally occur without the user's knowledge. The information can be used to tune WORKSIZE= and SYMSIZE= values for an application. However, the default WORKSIZE= and SYMSIZE= values should be appropriate in most applications.

Do not specify a very large value in the WORKSIZE= and SYMSIZE= options unless absolutely necessary. Many of the native functions and all of the DATA step functions used are dynamically loaded at execution time. If you use a large amount of the memory for symbolspace and workspace, there may not be enough remaining to load these functions, resulting in the error message

`Unable to load module` *module-name.*

Should you run into this problem, issue a SHOW SPACE command to examine current usage. You may be able to adjust the SYMSIZE= or WORKSIZE= values.

The amount of memory your system can provide depends on the capacity of your computer and on the products installed. The following techniques for efficient memory use are recommended when memory is at a premium:

- Free matrices as they are no longer needed using the FREE command.
- Store matrices you will need later in external library storage using the STORE command, and then FREE their values. You can restore the matrices later using the LOAD command. See Chapter 14, "Storage Features."
- Plan your work to use smaller matrices.

Accuracy

All numbers are stored and all arithmetic is done in double-precision. The algorithms used are generally very accurate numerically. However, when many operations are performed or when the matrices are ill-conditioned, matrix operations should be used in a numerically responsible way because numerical errors add up.

Error Diagnostics

When an error occurs, several lines of messages are printed. The error description, the operation being performed, and the line and column of the source for that operation are printed. The names of the operation's arguments are also printed. Matrix names beginning with a pound sign (#) or an asterisk (*) may appear; these are temporary names assigned by the IML procedure.

If an error occurs while you are in immediate mode, the operation is not completed and nothing is assigned to the result. If an error occurs while executing statements inside a module, a PAUSE command is automatically issued. You can correct the error and resume execution of module statements with a RESUME statement.

The most common errors are described below:

- referencing a matrix that has not been set to a value, that is, referencing a matrix that has no value associated with the matrix name
- making a subscripting error, that is, trying to refer to a row or column not present in the matrix
- performing an operation with nonconformable matrix arguments, for example, multiplying two matrices together that do not conform, or using a function that requires a special scalar or vector argument
- referencing a matrix that is not square for operations that require a square matrix (for example, INV, DET, or SOLVE)
- referencing a matrix that is not symmetric for operations that require a symmetric matrix (for example, GENEIG)
- referencing a matrix that is singular for operations that require a nonsingular matrix (for example, INV and SOLVE)
- referencing a matrix that is not positive definite or positive semidefinite for operations that require such matrices (for example, ROOT and SWEEP)
- not enough memory (see "Memory and Workspace" earlier in this chapter) to perform the computations and produce the result matrices.

These errors result from the actual dimensions or values of matrices and are caught only after a statement has begun to execute. Other errors, such as incorrect number of arguments or unbalanced parentheses, are syntax errors and resolution errors and are detected before the statement is executed.

Efficiency

The Interactive Matrix Language is an interpretive language executor that can be characterized as follows:

- efficient and inexpensive to compile
- inefficient and expensive for the number of operations executed
- efficient and inexpensive within each operation.

Therefore, you should try to substitute matrix operations for iterative loops. There is a high overhead involved in executing each instruction; however, within the instruction IML runs very efficiently.

Consider four methods of summing the elements of a matrix:

```
s=0;                       /* method 1 */
do i=1 to m;
   do j=1 to n;
      s=s+x[i,j];
   end;
end;
s=j[1,m]*x*j[n,1];         /* method 2 */
s=x[+,+];                  /* method 3 */
s=sum(x);                  /* method 4 */
```

Method 1 is the least efficient, method 2 is more efficient, method 3 is more efficient yet, and method 4 is the most efficient. The greatest advantage of using IML is reducing human programming labor.

Missing Values

An IML numeric element can have a special value called a *missing value* that indicates that the value is unknown or unspecified. (A matrix with missing values should not be confused with an empty or unvalued matrix, that is, a matrix with 0 rows and 0 columns.) A numeric matrix can have any mixture of missing and nonmissing values.

SAS/IML software supports missing values in a limited way. The operators listed below recognize missing values and propagate them. Most matrix operators and functions do not support missing values. For example, matrix multiplication or exponentiation involving a matrix with missing values is not meaningful. Also, the inverse of a matrix with missing values has no meaning.

Missing values are coded in the bit pattern of very large negative numbers, as an I.E.E.E. “NAN” code, or as a special string, depending on the host system.

In literals, a numeric missing value is specified as a single period. In data processing operations, you can add or delete missing values. All operations that move values around move missing values properly. The following arithmetic operators propagate missing values.

addition ($+$)	subtraction ($-$)
multiplication (#)	division (/)
maximum ($<>$)	minimum ($><$)
modulo (MOD)	exponentiation (##)

The comparison operators treat missing values as large negative numbers. The logical operators treat missing values as zeros. The operators SUM, SSQ, MAX, and MIN check for and exclude missing values.

The subscript reduction operators exclude missing values from calculations. If all of a row or column that is being reduced is missing, then the operator returns the result indicated in the table below.

Operator	**Result If All Missing**
addition ($+$)	0
multiplication (#)	1
maximum ($<>$)	large negative value
minimum ($><$)	large positive value
sum squares (##)	0
index maximum ($<:>$)	1
index minimum ($>:<$)	1
mean (:)	missing value

Also note that, unlike the SAS DATA step, IML does not distinguish between special and generic missing values; it treats all missing values alike.

Principles of Operation

This section presents various technical details on the operation of SAS/IML software. Statements in IML go through three phases:

- The parsing phase includes text acquisition, word scanning, recognition, syntactical analysis, and enqueuing on the statement queue. This is performed immediately as IML reads the statements.
- The resolution phase includes symbol resolution, label and transfer resolution, and function and call resolution. Symbol resolution connects the symbolic names in the statement with their descriptors in the symbol table. New symbols can be added or old ones recognized. Label and transfer resolution connects statements and references affecting the flow of control. This connects LINK and GOTO statements with labels; it connects IF with THEN and ELSE clauses; it connects DO with END. Function-call resolution identifies functions and call routines and loads them if necessary. Each reference is checked with respect to the number of arguments allowed. The resolution phase begins after a module definition is finished or a DO group is ended. For all other statements outside of any module or DO group, resolution begins immediately after parsing.

- The execution phase occurs when the statements are interpreted and executed. There are two levels of execution: statement and operation. Operation-level execution involves the evaluation of expressions within a statement.

Operation-Level Execution

Operations are executed from a chain of operation elements created at parse-time and resolved later. For each operation, the interpreter performs the following steps:

1. Prints a record of the operation if the FLOW option is on.
2. Looks at the operands to make sure they have values. Only certain special operators are allowed to tolerate operands that have not been set to a value. The interpreter checks whether any argument has character values.
3. Inspects the operator and gives control to the appropriate execution routine. A separate set of routines is invoked for character values.
4. Checks the operands to make sure they are valid for the operation. Then the routine allocates the result matrix and any extra workspace needed for intermediate calculations. Then the work is performed. Extra workspace is freed. A return code notifies IML if the operation was successful. If unsuccessful, it identifies the problem. Control is passed back to the interpreter.
5. Checks the return code. If the return code is nonzero, diagnostic routines are called to explain the problem to the user.
6. Associates the results with the result arguments in the symbol table. By keeping results out of the symbol table until this time, the operation does not destroy the previous value of the symbol if an error has occurred.
7. Prints the result if RESET PRINT or RESET PRINTALL is specified. The PRINTALL option prints intermediate results as well as end results.
8. Moves to the next operation.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ IML User's Guide, Version 8*, Cary, NC: SAS Institute Inc., 1999. 846 pp.

SAS/IML User's Guide, Version 8

ISBN 1–58025–553–1

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999