Chapter 2
Understanding the Language

Chapter Table of Contents

Chapter 2
Understanding the Language

Defining a Matrix

The fundamental data object on which all Interactive Matrix Language commands operate is a two-dimensional (row $\times$ column) numeric or character matrix. By their very nature, matrices are useful for representing data and efficient for working with data. Matrices have the following properties:

- Matrices can be either numeric or character. Elements of a numeric matrix are stored in double precision. Elements of a character matrix are character strings of equal length. The length can range from 1 to 32676 characters.
- Matrices are referred to by valid SAS names. Names can be from 1 to 8 characters long, beginning with a letter or underscore, and continuing with letters, numbers, and underscores.
- Matrices have dimension defined by the number of rows and columns.
- Matrices can contain elements that have missing values (see the section "Missing Values" on page 19 later in this chapter).

The dimension of a matrix is defined by the number of rows and columns it has. An $m \times n$ matrix has mn elements arranged in m rows and n columns. The following nomenclature is standard in this book:

- $1 \times n$ matrices are called *row vectors*.
- $m \times 1$ matrices are called *column vectors*.
- 1×1 matrices are called *scalars*.

Matrix Names and Literals

Matrix Names

A matrix is referred to by a valid SAS name. Names can be from 1 to 8 characters long, beginning with a letter or underscore and continuing with letters, numbers, and underscores. You associate a name with a matrix when you create or define the matrix. A matrix name exists independently of values. This means that at any time, you can change the values associated with a particular matrix name, change the dimension of the matrix, or even change its type (numeric or character).

Matrix Literals

A *matrix literal* is a matrix represented by its values. When you represent a matrix by a literal, you are simply specifying the values of each element of the matrix. A matrix literal can have a single element (a scalar) or have many elements arranged in a rectangular form (rows $\times$ columns). The matrix can be numeric (all elements are numeric) or character (all elements are character). The dimension of the matrix is automatically determined by the way you punctuate the values.

If there are multiple elements, use braces ({ }) to enclose the values and commas to separate the rows. Within the braces, values must be either all numeric or all character. If you use commas to create multiple rows, all rows must have the same number of elements (columns).

The values you input can be any of the following:

- a number, with or without decimal points, possibly in scientific notation (such as 1E−5)
- a character string. Character strings can be enclosed in either single quotes (’) or double quotes ("), but they do not necessarily need quotes. Quotes are required when there are no enclosing braces or when you want to preserve case, special characters, or blanks in the string. If the string has embedded quotes, you must double them (for example, WORD=’Can”t’). Special characters can be any of the following: ? = * : ( ).
- a period (.), representing a missing numeric value
- numbers in brackets ([ ]), representing repetition factors

Creating Matrices from Matrix Literals

Creating matrices using matrix literals is easy. You simply input the element values one at a time, usually inside of braces. Representing a matrix as a matrix literal is not the only way to create matrices. A matrix can also be created as a result of a function, a CALL statement, or an assignment statement. Below are some simple examples of matrix literals, some with a single element (scalars) and some with multiple elements. For more information on matrix literals, see Chapter 4, “Working with Matrices.”

Scalar Literals

The following examples define scalars as literals. These are examples of simple assignment statements, with the matrix name on the left-hand side of the equal sign and the value on the right. Notice that you do not need to use braces when there is only one element.

```
a=12;
a=. ;
a=’hi there’;
a="Hello";
```

Numeric Literals

Matrix literals with multiple elements have the elements enclosed in braces. Use commas to separate the rows of a matrix. For example, the statement

```
x={1 2 3 4 5 6};
```

assigns a row vector to the matrix **X**:

```
X
1          2          3          4          5          6
```

The statement

```
y={1,2,3,4,5};
```

assigns a column vector to the matrix **Y**:

```
Y
1
2
3
4
5
```

The statement

```
z={1 2, 3 4, 5 6};
```

assigns a 3×2 matrix literal to the matrix **Z**:

```
Z
1          2
3          4
5          6
```

The following assignment

```
w=3#z;
```

creates a matrix **W** that is three times the matrix **Z**:

```
W
 3          6
 9         12
15         18
```

Character Literals

You input a character matrix literal by entering character strings. If you do not use quotes, all characters are converted to uppercase. You must use either single or double quotes to preserve case or when blanks or special characters are present in the string. For character matrix literals, the length of the elements is determined from the longest element. Shorter strings are padded on the right with blanks. For example, the assignment of the literal

```
a={abc defg};
```

results in **A** being defined as a 1×2 character matrix with string length 4 (the length of the longer string).

```
A
ABC  DEFG
```

The assignment

```
a={'abc' 'DEFG'};
```

preserves the case of the elements, resulting in the matrix

```
A
abc  DEFG
```

Note that the string length is still 4.

Repetition Factors

A repetition factor can be placed in brackets before a literal element to have the element repeated. For example, the statement

```
answer={[2]  'Yes', [2]  'No'};
```

is equivalent to

```
answer={'Yes' 'Yes', 'No' 'No'};
```

and results in the matrix

```
ANSWER
Yes   Yes
No    No
```

Reassigning Values

You can assign new values to elements of a matrix at any time. The following statement creates a 2×3 numeric matrix named $\mathbf{A}$.

```
a={1 2 3, 6 5 4};
```

The statement

```
a={'Sales' 'Marketing' 'Administration'};
```

redefines the matrix $\mathbf{A}$ as a 1×3 character matrix.

Assignment Statements

Assignment statements create matrices by evaluating expressions and assigning the results to a matrix. The expressions can be composed of operators (for example, matrix multiplication) or functions (for example, matrix inversion) operating on matrices. Because of the nature of linear algebraic expressions, the resulting matrices automatically acquire appropriate characteristics and values. Assignment statements have the general form

result = *expression*;

where *result* is the name of the new matrix and *expression* is an expression that is evaluated, the results of which are assigned to the new matrix.

Functions as Expressions

Matrices can be created as a result of a function call. Scalar functions such as LOG or SQRT operate on each element of a matrix, while matrix functions such as INV or RANK operate on the entire matrix. For example, the statement

```
a=sqrt(b);
```

assigns the square root of each element of $\mathbf{B}$ to the corresponding element of $\mathbf{A}$.

The statement

```
y=inv(x);
```

calls the INV function to compute the inverse matrix of $\mathbf{X}$ and assign the results to $\mathbf{Y}$.

The statement

```
r=rank(x);
```

creates a matrix $\mathbf{R}$ with elements that are the ranks of the corresponding elements of $\mathbf{X}$.

Operators within Expressions

There are three types of operators that can be used in assignment statement expressions. Be sure that the matrices on which an operator acts are conformable to the

operation. For example, matrix multiplication requires that the number of columns of the left-hand matrix be equal to the number of rows of the right-hand matrix.

The three types of operators are as follows:

prefix operators	are placed in front of an operand ($-\mathbf{A}$).
infix operators	are placed between operands ($\mathbf{A} * \mathbf{B}$).
postfix operators	are placed after an operand ($\mathbf{A}'$).

All operators can work in a one-to-many or many-to-one manner; that is, they enable you to, for example, add a scalar to a matrix or divide a matrix by a scalar. The followingis an example of using operators in an assignment statement.

```
y=x#(x>0);
```

This assignment statement creates a matrix $\mathbf{Y}$ in which each negative element of the matrix $\mathbf{X}$ is replaced with zero. The statement actually has two expressions evaluated. The expression ($\mathbf{X}$>0) is a many-to-one operation that compares each element of $\mathbf{X}$ to zero and creates a temporary matrix of results; an element of the temporary matrix is 1 when the corresponding element of $\mathbf{X}$ is positive, and 0 otherwise. The original matrix $\mathbf{X}$ is then multiplied elementwise by the temporary matrix, resulting in the matrix $\mathbf{Y}$.

For a complete listing and explanation of operators, see Chapter 17, “Language Reference.”

Types of Statements

Statements in SAS/IML software can be classified into three general categories:

Control Statements
direct the flow of execution. For example, the IF-THEN/ELSE Statement conditionally controls statement execution.

Functions and **CALL Statements**
perform special tasks or user-defined operations. For example, the statement CALL: *GSTART* activates the SAS/IML graphics system.

Commands
perform special processing, such as setting options, displaying, and handling input/output. For example, the command RESET: *PRINT* turns on the automatic displaying option so that matrix results are displayed as you submit statements.

Control Statements

SAS/IML software has a set of statements for controlling program execution. Control statements direct the flow of execution of statements in IML. With them, you can define DO-groups and modules (also known as subroutines) and route execution of your program. Some control statements are described as follows.

Statements	Action
DO, END	group statements
iterative DO, END	define an iteration loop
GOTO, LINK	transfer control
IF-THEN/ELSE	routes execution conditionall
PAUSE	instructs a module to pause during execution
QUIT	ends a SAS/IML session
RESUME	instructs a module to resume execution
RETURN	returns from a LINK statement or a CALL module
RUN	executes a module
START, FINISH	define a module
STOP, ABORT	stop execution of an IML program

See Chapter 5, "Programming Statements," later in this book for more information on control statements.

Functions

The general form of a function is

$$result = FUNCTION(arguments);$$

where *arguments* can be matrix names, matrix literals, or expressions. Functions always return a single result (whereas subroutines can return multiple results or no result). If a function returns a character result, the matrix to hold the result is allocated with a string length equal to the longest element, and all shorter elements are padded with blanks.

Categories of Functions

Functions fall into the following six categories:

matrix inquiry functions
return information about a matrix. For example, the ANY function returns a value of 1 if any of the elements of the argument matrix are nonzero.

scalar functions
operate on each element of the matrix argument. For example, the ABS function returns a matrix with elements that are the absolute values of the corresponding elements of the argument matrix.

summary functions
return summary statistics based on all elements of the matrix argument. For example, the SSQ function returns the sum of squares of all elements of the argument matrix.

matrix arithmetic functions
perform matrix algebraic operations on the argument. For example, the TRACE function returns the trace of the argument matrix.

matrix reshaping functions
manipulate the matrix argument and return a reshaped matrix. For example, the DIAG function returns a matrix with diagonal elements that are equal to the diagonal elements of a square argument matrix. All off-diagonal elements are zero.

linear algebra and statistical functions
perform linear algebraic functions on the matrix argument. For example, the GINV function returns the matrix that is the generalized inverse of the argument matrix.

Exceptions to the SAS DATA Step

SAS/IML software supports most functions supported in the SAS DATA step. These functions all accept matrix arguments, and the result has the same dimension as the argument. (See Appendix 1 for a list of these functions.) The following functions are **not** supported by SAS/IML software:

DIFn	HBOUND	LAGn	PUT
DIM	INPUT	LBOUND	

The following functions are implemented differently in SAS/IML software. (See Chapter 17, "Language Reference," for descriptions.)

MAX	RANK	SOUND	SUBSTR
MIN	REPEAT	SSQ	SUM

The random number functions, UNIFORM and NORMAL, are built-in and produce the same streams as the RANUNI and RANNOR functions, respectively, of the DATA step. For example, to create a 10×1 vector of random numbers, use

```
x=uniform(repeat(0,10,1));
```

Also, SAS/IML software does not support the OF clause of the SAS DATA step. For example, the statement

```
a=mean(of x1-x10); /* invalid in IML */
```

cannot be interpreted properly in IML. The term (X1-X10) would be interpreted as subtraction of the two matrix arguments rather than its DATA step meaning, "X1 through X10."

CALL Statements and Subroutines

CALL statements invoke a subroutine to perform calculations, operations, or a service. CALL statements are often used in place of functions when the operation returns multiple results or, in some cases, no result. The general form of the CALL statement is

CALL *SUBROUTINE arguments* **;**

where *arguments* can be matrix names, matrix literals, or expressions. If you specify several arguments, use commas to separate them. Also, when using arguments for output results, always use variable names rather than expressions or literals.

Creating Matrices with CALL Statements

Matrices are created whenever a CALL statement returns one or more result matrices. For example, the statement

```
call eigen(val,vec,t);
```

returns two matrices (vectors), **VAL** and **VEC**, containing the eigenvalues and eigenvectors, respectively, of the symmetric matrix **T**.

You can program your own subroutine using the START and FINISH statements to define a module. You can then execute the module with a CALL statement or a RUN statement. For example, the following statements define a module named MYMOD that returns matrices containing the square root and log of each element of the argument matrix:

```
start mymod(a,b,c);
   a=sqrt(c);
   b=log(c);
finish;
run mymod(s,l,x);
```

Execution of the module statements create matrices **S** and **L**, containing the square roots and logs, respectively, of the elements of **X**.

Performing Services

You can use CALL statements to perform special services, such as managing SAS data sets or accessing the graphics system. For example, the statement

```
call delete(mydata);
```

deletes the SAS data set named MYDATA.

The statements

```
call gstart;
call gopen;
call gpoint(x,y);
call gshow;
```

activate the graphics system (CALL GSTART), open a new graphics segment (CALL GOPEN), produce a scatter plot of points (CALL GPOINT), and display the graph (CALL GSHOW).

Commands

Commands are used to perform specific system actions, such as storing and loading matrices and modules, or to perform special data processing requests. The following is a list of some commands and the actions they perform.

Command	Action
FREE	frees a matrix of its values and increases available space
LOAD	loads a matrix or module from the storage library
MATTRIB	associates printing attributes with matrices
PRINT	prints a matrix or message
RESET	sets various system options
REMOVE	removes a matrix or module from library storage
SHOW	requests that system information be displayed
STORE	stores a matrix or module in the storage library

These commands play an important role in SAS/IML software. With them, for example, you can control displayed output (with RESET PRINT, RESET NOPRINT, or MATTRIB) or get system information (with SHOW SPACE, SHOW STORAGE, or SHOW ALL).

If you are running short on available space, you can use commands to store matrices in the storage library, free the matrices of their values, and load them back later when you need them again, as shown in the following example.

Throughout this session, the right angle brackets (>) indicate statements that you submit; responses from IML follow. First, invoke the procedure by entering PROC IML at the input prompt. Then, create matrices **A** and **B** as matrix literals.

```
> proc iml;

  IML Ready

> a={1 2 3, 4 5 6, 7 8 9};
> b={2 2 2};
```

List the names and attributes of all of your matrices with the SHOW NAMES command.

```
> show names;

  A             3 rows      3 cols num      8
  B             1 row       3 cols num      8
  Number of symbols = 2  (includes those without values)
```

Store these matrices in library storage with the STORE command, and release the space with the FREE command. To list the matrices and modules in library storage, use the SHOW STORAGE command.

```
> store a b;
> free a b;
> show storage;

  Contents of storage = SASUSER.IMLSTOR
  Matrices:
   A        B

  Modules:
```

The output from the SHOW STORAGE statement indicates that you have two matrices in storage. Because you have not stored any modules in this session, there are no modules listed in storage. Return these matrices from the storage library with the LOAD command. (See Chapter 14, "Storage Features," for details about storage.)

```
> load a b;
```

End the session with the QUIT command.

```
> quit;

  Exiting IML
```

Data Management Commands

SAS/IML software has many data management commands that enable you to manage your SAS data sets from within the SAS/IML environment. These data management commands operate on SAS data sets. There are also commands for accessing external files. The following is a list of some commands and the actions they perform.

Command	Action
APPEND	adds records to an output SAS data set
CLOSE	closes a SAS data set
CREATE	creates a new SAS data set
DELETE	deletes records in an output SAS data set
EDIT	reads from or writes to an existing SAS data set
FIND	finds records that meet some condition
LIST	lists records
PURGE	purges records marked for deletion
READ	reads records from a SAS data set into IML variables
SETIN	makes a SAS data set the current input data set
SETOUT	makes a SAS data set the current output data set
SORT	sorts a SAS data set
USE	opens an existing SAS data set for read access

These commands can be used to perform any necessary data management functions. For example, you can read observations from a SAS data set into a target matrix with the USE or EDIT command. You can edit a SAS data set, appending or deleting records. If you have generated data in a matrix, you can output the data to a SAS data set with the APPEND or CREATE command. See Chapter 6, "Working with SAS Data Sets," and Chapter 7, "File Access," for more information on these commands.

Missing Values

With SAS/IML software, a numeric element can have a special value called a *missing value* that indicates that the value is unknown or unspecified. Such missing values are coded, for logical comparison purposes, in the bit pattern of very large negative numbers. A numeric matrix can have any mixture of missing and nonmissing values.

A matrix with missing values should not be confused with an empty or unvalued matrix, that is, a matrix with zero rows and zero columns.

In matrix literals, a numeric missing value is specified as a single period. In data processing operations involving a SAS data set, you can append or delete missing values. All operations that move values move missing values properly.

SAS/IML software supports missing values in a limited way, however. Most matrix operators and functions do not support missing values. For example, matrix multiplication involving a matrix with missing values is not meaningful. Also, the inverse of a matrix with missing values has no meaning. Performing matrix operations such as these on matrices that have missing values can result in inconsistencies, depending on the host environment.

See Chapter 4, "Working with Matrices," and Chapter 16, "Further Notes," for more details on missing values.

Summary

In this chapter, you were introduced to the fundamentals of the SAS/IML language. The basic data element, the matrix, was defined, and you learned several ways to create matrices: the matrix literal, CALL statements that return matrix results, and assignment statements.

You were introduced to the types of statements with which you can program: commands, control statements for iterative programming and module definition, functions, and CALL subroutines.

Chapter 3, "Tutorial: A Module for Linear Regression," offers an introductory tutorial that demonstrates using SAS/IML software to build and execute a module.

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