CHAPTER 11. Matrix Algebra with SAS/IML Software

a. Introductory material about matrices
b. IML assignment statements and operators
c. Subscripts
d. IML functions
e. Working with SAS data sets
f. Examples:
    1. Solving a system of equations
    2. Linear regression

- IML stands for Interactive Matrix Language.

- SAS/IML software is part of the SAS System.
  - You can access SAS data sets or external files.
  - You can edit existing SAS data sets or create new SAS data sets.

- The fundamental data element in SAS/IML is a data matrix.

- SAS/IML is a programming language.
  - A wide range of subroutines is available.
  - You have access to many operators and functions.
  - A complete set of control statements is available.
  - You can define your own functions or subroutines using SAS/IML modules.

- SAS/IML uses operators and functions that apply to an entire matrix.

- SAS/IML is interactive.
  - Commands can be executed as entered, or
  - Commands can be collected in a module and executed later.

- SAS/IML processes data.
  - You can read all observations or conditionally select observations from a SAS data set into a matrix.
  - You can create a new SAS data set or edit or append observations to an existing SAS data set.

SAMPLE IML SESSION

```
21   PROC IML;
NOTE: IML Ready
22   RESET PRINT;
23
24   a=3;
25
26   b={1 2 3};
27
28   c={1, 2, 3};
29
30   d={1 2 3, 4 5 6};
31
32   QUIT;
NOTE: Exiting IML.
NOTE: PROCEDURE IML used:
      real time           0.04 seconds
      cpu time            0.04 seconds

 --------------------------------------------------------------

A      1 row       1 col     (numeric)


        3

B      1 row       3 cols    (numeric)


        1         2         3

C      3 rows      1 col     (numeric)


        1
        2
        3

D      2 rows      3 cols    (numeric)


        1         2         3
        4         5         6
```

DEFINING A MATRIX

- The fundamental data object on which all IML commands operate is a two-dimensional numeric or character matrix.

- Matrices in SAS have the following properties:

  - Matrices can be either character or numeric.
  - Elements in a numeric matrix are stored in double precision.
  - Elements of a character matrix are character strings of equal length.
  - Matrices are referred to by valid SAS names.
  - Matrices have dimension defined by the number of rows and columns.
  - Matrices can contain elements that have missing values.

- The dimension of a matrix is defined by the number of rows and columns it has. An m X n matrix has mn elements arranged in m rows and n columns.

  1 X n matrices are called row vectors.

  m X 1 matrices are called column vectors.

  1 X 1 matrices are called scalers.

ROW VECTOR 1 X N

```
 R            1 row        3 cols    (numeric)

              1          2          3
```

COLUMN VECTOR M X 1

```
 C             3 rows       1 col     (numeric)

                             1
                             2
                             3
```

SCALAR 1 X 1

```
  S            1 row        1 col     (numeric)

                            3
```

M X N MATRIX

```
  M            3 rows       3 cols    (numeric)

                 1          2          3
                 4          5          6
                 7          8          9
```

MATRIX NAMES and MATRIX LITERALS

A) Matrix Names

- A matrix is referred to by a valid SAS name.
- Naming convention rules are the same as in base SAS except that names can only be up to 8 characters long.
- A name is associated with a matrix when the matrix is created.
- The type (character or numeric), dimension, or values of a matrix can be changed at any time.

B) Matrix Literals

- When you represent a matrix by a literal, you specify the values of each element of the matrix.
- Matrix literals can have a single element or many elements arranged in rectangular form.
- 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.
- All rows must have the same number of elements.

C) Scaler Literals

```
a=12 ;
a=. ;
a='bios 511' ;
a="BIOS511" ;
```

D) Numeric Literals

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

assigns a row vector to the matrix X

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

assigns a column vector to the matrix Y

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

assigns a 3X2 matrix literal to the matrix Z

E) Character Literals

- Are input by entering character strings.
- If you don't use quotes, all elements are converted to upper case.
- You must use single or double quotes to preserve case or when blanks or special characters are in the string.
- The length of the elements is determined by the length of the longest string.
- Shorter strings are padded on the right with blanks.

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

creates A, a 1X2 matrix, with string length 4

```
                A
                ABC   DEFG
```

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

preserves the case of the elements

```
                A
                abc   DEFG
```

F) Reassigning Values

You can reassign values to a matrix at any time.

The statement below creates a 2X3 numeric matrix named **A** .

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

The statement

```
    a={'Gonzo' 'Piggy' } ;
```

redefines matrix **A** as a 1X2 character matrix.

Scaler Literals Example

```
12   TITLE 'Scaler Literals';
13   PROC IML;
NOTE: IML Ready
14   RESET PRINT;
15
16   a = 12;
17   b = .;
18   c = 'bios 511';
19   d = "BIOS511";
20
21   QUIT;
NOTE: Exiting IML.
NOTE: PROCEDURE IML used:
      real time           0.05 seconds
      cpu time            0.05 seconds

--------------------------------------------------------------

Scaler Literals

A      1 row       1 col     (numeric)


       12

B      1 row       1 col     (numeric)


        .

C      1 row       1 col     (character, size 8)


bios 511

D      1 row       1 col     (character, size 7)


BIOS511
```

Numeric Literals Example

```
22   TITLE 'Numeric Literals';
23   PROC IML;
NOTE: IML Ready
24   RESET PRINT;
25
26   x = {1 2 3 4 5 6};
27   y = {1,2,3,4,5,6};
28   z = {1 2, 3 4, 5 6};
29
30   QUIT;
NOTE: Exiting IML.
NOTE: PROCEDURE IML used:
      real time           0.04 seconds
      cpu time            0.04 seconds

--------------------------------------------------------------

Numeric Literals

X       1 row       6 cols    (numeric)


        1         2         3         4         5         6

Y       6 rows      1 col     (numeric)


        1
        2
        3
        4
        5
        6

Z       3 rows      2 cols    (numeric)


        1         2
        3         4
        5         6
```

Character Literals Example

```
31   TITLE 'Character Literals';
32   PROC IML;
NOTE: IML Ready
33   RESET PRINT;
34
35   a = {abc defg};
36   b = {'abc' 'defg'};
37   b = {1 2 3, 4 5 6};
38
39   QUIT;
NOTE: Exiting IML.
NOTE: PROCEDURE IML used:
      real time           0.04 seconds
      cpu time            0.04 seconds

-------------------------------------------------------------

Character Literals

A      1 row       2 cols    (character, size 4)


ABC  DEFG

B      1 row       2 cols    (character, size 4)


abc  defg

B      2 rows      3 cols    (numeric)


        1         2         3
        4         5         6
```

IML: ASSIGNMENT STATEMENTS

Assignment statements create matrices by evaluating expressions and assigning results to a matrix.

The expressions can be composed of operators or functions.

Assignments statements have the general form

```
result = expression ;
```

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

OPERATORS

There are three general types of operators used in matrix expressions:

prefix operators are placed in front of operands . For example, -A uses the sign reverse prefix operator in front of the operand A to reverse the sign of each element of A.

infix operators are placed between operands. For example, A+B uses the addition infix operator (+).

postfix operators are placed after an operand. For example, A` uses the transpose postfix operator after the operand A to transpose A.

Table A1.1 Operators

Operator	Symbol	Syntax type	Data type
Sign reverse	-	Prefix	Num
Addition	+	Infix	Num
Subtraction	-	Infix	Num
Index creation	:	Infix	Num
Matrix multiplication	*	Infix	Num
Elementwise multiplication	#	Infix	Num
Matrix power	**	Infix	Num
Elementwise power	##	Infix	Num
Division	/	Infix	Num
Horizontal concatenation	||	Infix	Both
Vertical concatenation	//	Infix	Both
Element maximum	<>	Infix	Both
Element minimum	><	Infix	Both
Logical AND	&	Infix	Num
Logical OR	|	Infix	Num
Logical NOT	^	Prefix	Num
Less than	<	Infix	Both
Greater than	>	Infix	Both
Equal to	=	Infix	Both
Less than or equal to	<=	Infix	Both
Greater than or equal to	>=	Infix	Both
Not equal to	^=	Infix	Both
Transpose	`	Postfix	Both
Subscripts	[ ]	Postfix	both

Table A1.2 Subscript Reduction Operators

Operator	Action
+	Addition
#	Multiplication
<>	Maximum
><	Minimum
<:>	Index of maximum
>:<	Index of minimum
:	Mean
##	Sum of squares

Table A1.3 Operator Precedence

Priority Group	**Operators**
I (highest)	^ ` subscripts -(prefix) ## **
II	* # <> >< / @
III	+ -
IV	|| // :
V	< <= > >= = ^=
VI	&
VII (lowest)	|

A higher priority means that those operators are evaluated first. Thus, Group I operators are evaluated before Group II operators.

If neighboring operators in an expression have equal precedence, the expression is evaluated from left to right, except for Group I operators, which are evaluated from right to left.

Examples:

Evaluated from left to right: a = x / y / z; [evaluated as (x / y) / z]

Evaluated from right to left: z = -x**2; [evaluated as –(x**2)]

EXAMPLES:

```
LET  A=  2   2   and   B=   4   5
         3   4              1   0


A + B yields           6  7   addition (elementwise)
                       4  4


A # B yields           8 10   elementwise
                       3  0   multiplication


A * B yields          10 10   matrix
                      16 15   multiplication


A ## 2 yields          4  4   element power
                       9 16   operator


A <> B yields          4  5   element maximum
                       3  4   operator


A <= B yields          1  1   less than or
                       0  0   equal to operator


A`                     2  3   transpose operator
                       2  4


A || B  yields         2  2  4  5       horizontal
                       3  4  1  0       concatenation


A // B yields          2  2  vertical
                       3  4  concatenation
                       4  5
                       1  0
```

Operators Example

```
14   PROC IML;
NOTE: IML Ready
15   RESET PRINT;
16
17   a = {2 2, 3 4};
18   b = {4 5, 1 0};
19   c = -b;
20   d = a + b;
21   e = a`;
22
23   QUIT;
NOTE: Exiting IML.
NOTE: PROCEDURE IML used:
      real time           0.07 seconds

--------------------------------------------------------------

Operators

A      2 rows      2 cols    (numeric)

        2         2
        3         4

B      2 rows      2 cols    (numeric)

        4         5
        1         0

C      2 rows      2 cols    (numeric)

       -4        -5
       -1         0

D      2 rows      2 cols    (numeric)

        6         7
        4         4

E      2 rows      2 cols    (numeric)

        2         3
        2         4
```

IML: SUBSCRIPTS

Subscripts are special postfix operators placed in square brackets([ ]) after a matrix operand. Subscript operators have the general form

```
operand[row,column]
```

where

operand	is usually a matrix name
row	is an expression selecting 1 or more rows
column	is an expression selecting 1 or more columns

Subscripts are used to:

- refer to a single element of a matrix
- refer to an entire row or column of a matrix
- refer to a submatrix within a matrix
- perform a reduction across rows or columns of a matrix.

SELECTING A SINGLE ELEMENT

You can select a single element in two ways. You can use two subscripts (row,column), or you can use one subscript to look for the element across and down the rows (this is called row-major).

For example, if

```
 A =  1  2  3
      4  5  6
      7  8  9
```

then

```
 B = A[2,3] ;
```

yields 6 for B.

You can also look for an element across and down the rows. In this case you would refer to the element as the sixth element of A.

```
B=A[6] ;
```

SELECTING A ROW OR COLUMN

To select an entire row or column, write the subscript with the row or column number, omitting the other subscript but not the comma.

To select row 1 of matrix A:

```
   C=A[1,] ;                C
                            1  2  3
```

To select column 2 of A:

```
   D=A[,2] ;                D
                            2
                            5
                            8
```

SUBMATRICES

You can refer to a submatrix by the specific rows and columns that you want. Include within the brackets the row you want, a comma, and the columns you want.

To create a submatrix consisting of the first and third rows and the first and second columns of A:

```
E=A[{1 3},{1 2}] ;          E
                            1  2
                            7  8
```

You can use the index creation operator (:) to refer to successive rows or columns. For example, to create a submatrix consisting of rows 1-3 and columns 1-2 from B where :

```
    B =  1  2  3  4
         5  6  7  8
         9  10 11 12
```

use `F = B[1:3,1:2];` which yields

```
F

1  2
5  6
9  10
```

SUBSCRIPT ASSIGNMENT

You can assign values into a matrix using subscripts to refer to the element or submatrix. For example to change the value in the first row, second column of A from 2 to 30:

```
A[1,2]=30;
```

To change the values in column 3 of A:

```
A[,3]={20 30 40} ;          A
                            1  2  20
                            4  5  30
                            7  8  40
```

SUBSCRIPT REDUCTION OPERATORS

You can use subscript reduction operators to perform operations across all rows or columns. Using these operators will result in a matrix of reduced dimensions. For example, subscript reduction operators can be used to obtain column sums, row sums, or column or row means. To get column sums of the matrix X (that is, sum across the rows, which reduces the row dimension to 1), specify X[+,]. The first subscript (+) indicates summation reduction takes place across the rows. Omitting the column subscript leaves the column dimension unchanged.

These operators can be used to reduce rows, columns, or both. When both rows and columns are reduced, rows are reduced first. Table A1.2 (p. 12) lists the eight operators for subscript reduction.

Subscript Reduction Operators Examples

```
24   PROC IML;
NOTE: IML Ready
25   RESET PRINT;
26
27   a = {0 1 2, 5 4 3, 7 6 8};
28   b = a[,+];
29   c = a[,:];
30   d = a[+,];
31   e = a[:,];
32
33   QUIT;
NOTE: Exiting IML.
NOTE: PROCEDURE IML used:
      real time           0.05 seconds

--------------------------------------------------------------

Subscript Reduction Operators

A      3 rows      3 cols    (numeric)

        0         1         2
        5         4         3
        7         6         8

B      3 rows      1 col     (numeric)

        3
       12
       21

C      3 rows      1 col     (numeric)

        1
        4
        7

D      1 row       3 cols    (numeric)

       12        11        13

E      1 row       3 cols    (numeric)

        4 3.6666667 4.3333333
```

FUNCTIONS AS EXPRESSIONS

Matrices can also be created as a result of a function call. Scalar functions such as LOG or SQRT operate on each element of the matrix, while matrix function such as INV or RANK operate on the entire matrix. The general form of a function is:

```
result = FUNCTION(arguments) ;
```

For example:

```
a = SQRT(b) ;
```

assigns the square root of each element of B to the corresponding element of A.

MATRIX GENERATING FUNCTIONS

THE BLOCK FUNCTION

```
  BLOCK(matrix1,matrix2,...) ;
```

creates a block diagonal matrix from the argument matrices

```
A             2 rows      2 cols    (numeric)

        1         1
        1         1


B             2 rows      2 cols    (numeric)

        2         2
        2         2
```

C = BLOCK(A,B) ; RESULTS IN THE MATRIX

```
C             4 rows      4 cols    (numeric)

        1         1         0         0
        1         1         0         0
        0         0         2         2
        0         0         2         2
```

THE J FUNCTION

```
  J(nrow,rcol,value) ;
```

creates a matrix with *nrow* rows, *ncol* columns, and all element values equal to *value*

C = J(2,2,1) ; RESULTS IN THE MATRIX

```
C             2 rows      2 cols    (numeric)

        1         1
        1         1
```

THE I FUNCTION

```
  I(dimension) ;
```

generates an identity matrix with *dimension* rows and *dimension* columns.

C = I(3) ; RESULTS IN THE MATRIX

```
C      3 rows      3 cols    (numeric)

        1         0         0
        0         1         0
        0         0         1
```

OTHER USEFUL FUNCTIONS

THE NCOL FUNCTION

```
  NCOL(matrix) ;
```

Returns a scalar containing the number of columns in a matrix.

For example, to let B contain the number of columns in A, use the statement

```
   B = NCOL(A) ;
```

THE NROW FUNCTION

```
  NROW(matrix) ;
```

Returns a scalar containing the number of rows in a matrix.

For example, to let B contain the number of rows in A, use the statement

```
   B = NROW(A) ;
```

THE ABS FUNCTION

```
  ABS(matrix) ;
```

Returns a matrix containing the absolute value of every element in the input matrix, where *matrix* is a numeric matrix or literal.

For example, to let C contain the absolute value of every element in A, use

```
   C = ABS(A) ;
```

THE SQRT FUNCTION

```
  SQRT(matrix) ;
```

Returns a matrix containing the square root of every element in the input matrix, where *matrix* is a numeric matrix or literal.

For example, to let C contain the square root of every element in A, use

```
   C = SQRT(A) ;
```

THE EXP FUNCTION

```
  EXP(matrix) ;
```

Returns a matrix containing the exponential function of every element in the input matrix, where *matrix* is a numeric matrix or literal. The exponential is the natural number *e* raised to the indicated power.

```
 B = {2 3 4} ;

 A = EXP(b) ;   returns the matrix

        A      1 row         3 col     (numeric)

           7.3890561   20.085537   54.59815
```

THE LOG FUNCTION

```
  LOG(matrix) ;
```

Returns a matrix containing the natural (base e) logarithm of every element in the input matrix, where *matrix* is a numeric matrix or literal.

For example, to let C contain the natural log of every element in A, use

```
  C = LOG(A) ;
```

THE MAX FUNCTION

```
  MAX(matrix1,matrix2, ...matrix15) ;
```

Returns a scalar that is the largest element in all arguments. There can be as many as 15 argument matrices. The input matrices can be numeric, character, or literal. The function checks for missing numeric values and does not include them in the result.

An example of how to use the MAX function is

```
  C = MAX(A,B) ;
```

Note 1: There is also a parallel MIN function.

Note 2: If you want to find the elementwise maximums of the corresponding elements of two matrices, use the maximum operator (<>).

THE DET FUNCTION

```
  DET(matrix) ;
```

Returns a scalar containing the determinant of a square matrix, where *matrix* is a square numeric matrix or literal.

An example of how to use the DET function is C = DET(A) ;

```
Determinant reminder: If M = {a b,
                              c d} the determinant of M is ad – bc.
```

THE INV FUNCTION

```
  INV(matrix) ;
```

Returns the inverse of a matrix, where matrix is a square and nonsingular matrix.

An example of how to use the INV function is

```
  C = INV(A) ;
```

Note: C * A would then equal the identity matrix I (square matrix of all 0’s except for 1’s on upper left to lower right diagonal).

THE DIAG FUNCTION

```
  DIAG(matrix) ;
```

where *matrix* can either be a numeric square matrix or a vector.

If matrix is a square matrix, the DIAG function creates a matrix with diagonal elements equal to the corresponding diagonal elements. All off-diagonal elements in the new matrix are zeros.

If matrix is a vector, the DIAG function creates a matrix whose diagonal elements are the values of the vector. All off-diagonal elements are zeros.

For example,

```
A = {4 3,
     2 1} ;

C = DIAG(A) ;  results in

    C      2 rows      2 cols    (numeric)


                 4         0
                 0         1


B = {1 2 3} ;
D = DIAG(B) ;   results in


    D      3 rows      3 cols    (numeric)

             1         0         0
             0         2         0
             0         0         3
```

THE VECDIAG FUNCTION

```
  VECDIAG(matrix) ;
```

returns a column vector whose elements are the main diagonal elements of matrix, where *matrix* is a square numeric matrix.

For example

```
A = {4 3,
     2 1} ;

C = VECDIAG(A) ;  results in

    C      2 rows      1 cols    (numeric)

                        4
                        1
```

Functions Example

```
PROC LOGISTIC DATA=bios511.v2 COVOUT
   OUTEST=out(DROP=_link_ _status_ _type_ _name_ _lnlike_) NOPRINT ;
   MODEL smoker = sysba chol trig alcohol;
RUN;

PROC PRINT DATA=out;
TITLE 'OUT Data Set';
FORMAT intercept sysba chol trig alcohol 8.5;
RUN;

TITLE 'IML Output';
PROC IML;
   RESET PRINT FW=8 FUZZ=.00001;
   USE out;
   READ ALL INTO vb;  /* betas/variance covariance matrix */
   nvb = NROW(vb);    /* number of rows in VB            */
   beta = vb[1,];     /* row vector of betas             */
   beta = beta`;      /* convert to column vector of betas*/
   v = vb[2:nvb,];    /* variance covariance matrix      */
   v = VECDIAG(v);    /* column vector of variances      */
   s = SQRT(v);       /* column vector of standard errors */
   l = beta - (1.96#s) ;  /* lower 95% CI                */
   h = beta + (1.96#s) ;  /* upper 95% CI                */
   lbh = l || beta || h ; /* lower CI, betas, upper CI    */
QUIT;
```

```
OUT Data Set

Obs    Intercept       SYSBA        CHOL        TRIG     ALCOHOL

 1       2.10334     0.01019    -0.01004     0.00023    -0.02835
 2       2.94187    -0.01546    -0.00389     0.00001    -0.00135
 3      -0.01546     0.00014    -0.00001    -0.00000    -0.00002
 4      -0.00389    -0.00001     0.00002    -0.00000     0.00001
 5       0.00001    -0.00000    -0.00000     0.00000    -0.00000
 6      -0.00135    -0.00002     0.00001    -0.00000     0.00016
```

IML Output

```
VB      6 rows      5 cols    (numeric)

2.103345 0.010191 -0.01004 0.000232 -0.02835
2.941872 -0.01546 -0.00389        0 -0.00135
-0.01546 0.000136        0        0 -0.00002
-0.00389        0  0.00002        0        0
       0        0        0        0        0
-0.00135 -0.00002        0        0 0.000157


NVB      1 row       1 col     (numeric)

        6


BETA      1 row       5 cols    (numeric)

2.103345 0.010191 -0.01004 0.000232 -0.02835


BETA      5 rows      1 col     (numeric)

2.103345
0.010191
-0.01004
0.000232
-0.02835


V      5 rows      5 cols    (numeric)

2.941872 -0.01546 -0.00389        0 -0.00135
-0.01546 0.000136        0        0 -0.00002
-0.00389        0  0.00002        0        0
       0        0        0        0        0
-0.00135 -0.00002        0        0 0.000157
```

IML Output

```
V      5 rows      1 col     (numeric)

2.941872
0.000136
 0.00002
       0
0.000157


S      5 rows      1 col     (numeric)

1.715189
0.011648
0.004481
0.002074
0.012515


L      5 rows      1 col     (numeric)

-1.25842
-0.01264
-0.01882
-0.00383
-0.05288


H      5 rows      1 col     (numeric)

5.465114
0.033021
-0.00125
0.004296
-0.00382


LBH      5 rows      3 cols    (numeric)

-1.25842 2.103345 5.465114
-0.01264 0.010191 0.033021
-0.01882 -0.01004 -0.00125
-0.00383 0.000232 0.004296
-0.05288 -0.02835 -0.00382
```

WORKING WITH SAS DATA SETS

- SAS/IML software can pass data from SAS data sets to matrices and from matrices to SAS data sets.
- You can create matrices from the variables and observations in several ways. You can create a column vector for each data set variable, or you can create a matrix where columns correspond to data set variables.
- You can use all observations in a data set or a subset of them.
- You can create a SAS data set from a matrix. The columns correspond to data set variables and the rows correspond to observations.

OPENING A SAS DATA SET

Before you can access a SAS data set, you must first open it. There are three ways to open a SAS data set:

- To read from an existing SAS data set, submit a **USE** command to open it. The general form of the **USE** statement is:

  USE sas_data_set <VAR operand> <WHERE expression>;

- To read and write to an existing SAS data set, use the **EDIT** statement. The general form of the **EDIT** statement is:

  EDIT sas_data_set <VAR operand> <WHERE expression>;

- To create a new SAS data set, use the **CREATE** statement. The general form of the **CREATE** statement is:

  CREATE sas_data_set <VAR operand>;

  CREATE sas_data_set FROM from_name

          <[COLNAME=column_name ROWNAME=row_name}> ;

  Use the **APPEND** statement to place observations into the newly created data set. If you don't use the **APPEND** statement, the new data set will have no records.

READING OBSERVATIONS FROM A SAS DATA SET

- Transferring data from a SAS data set to a matrix is done with the **READ** statement.
- The SAS data set that you want to read from must already be open**.**

- You can open a SAS data set with the USE or EDIT statement.

- The general form of the **READ** statement is:

  READ <range> <VAR operand> <WHERE expression> <INTO name> ;

where

range	specifies a range of observations
operand	selects a set of variables
expression	is a true/false expression
name	names a target matrix for the data

USING THE READ STATEMENT WITH THE VAR CLAUSE

- Use the READ statement with the VAR clause to read variables from the current SAS data set into column vectors of the VAR clause.

- Each variable in the VAR clause becomes a column vector with the same name as the variable in the SAS data set.

- The number of observations is equal to the number of observations processed, depending on the range specification and the WHERE clause.

- For example, to read the variables AGE, HEIGHT, and WEIGHT for all observations in CLASS, use the statements

  USE bios511.class ;
  READ all **VAR**{age height weight} ;

  The READ statement above creates three numeric vectors AGE, HEIGHT, and WEIGHT. The two statements above can also be written as:

  USE bios511.class **VAR**{age height weight} ;
  READ all ;

USING THE READ STATEMENT WITH THE VAR & INTO CLAUSES

- You can use the READ statement with the VAR and INTO clauses to read the variables listed in the VAR clause into a single matrix named in the INTO clause.

- Each variable in the VAR clause becomes a column in the target matrix. If there are p variables in the VAR clause and n observations processed, the target matrix in the INTO clause is an nXp matrix.

- The following creates a matrix X containing all numeric variables in the CLASS data set.

  USE bios511.class ;
  READ all **VAR _**num**_ INTO** X ;

- You can specify ROWNAME= and COLNAME= matrices as part of the INTO clause.
  - The COLNAME= matrix specifies the name of a new character matrix to be created. This COLNAME= matrix is created in addition to the target matrix of the INTO clause and contains variable names from the input data set corresponding to columns of the target matrix. The COLNAME= matrix has dimension 1 ×*nvar*, where *nvar* is the number of variables contributing to the target matrix.
  - The ROWNAME= option specifies the name of a character variable in the input data set. The values of this variable are put in a character matrix with the same name as the variable. This matrix has the dimension *nobs*×1, where *nobs* is the number of observations in the range of the READ statement.

USING THE READ STATEMENT WITH THE WHERE CLAUSE

You can use a **WHERE** clause to conditionally select observations from within the specified range. If you want to create a matrix FEMALE containing AGE, HEIGHT, and WEIGHT for females only:

USE bios511.class ;
READ all **VAR _**num**_ INTO** female **WHERE**(sex="F");

CREATING A SAS DATA SET FROM A MATRIX

- The **CREATE** and **APPEND** statements can be used to create a SAS data set from a matrix.

- The columns of the matrix become the data set variables, and the rows of the matrix become the observations.

- An nXm matrix creates a SAS data set with m variables and n observations.

- The **CREATE** statement opens the new SAS data set for both input and output, and the **APPEND** statement writes to the SAS data set.

USING THE CREATE STATEMENT WITH THE FROM OPTION

You can create a SAS data set from a matrix using the **CREATE** statement with the **FROM** option. This form of the CREATE statement is:

CREATE sas_data_set **FROM** matrix
<[COLNAME=column_name ROWNAME=row_name]> ;

where

sas_data_set	names the new SAS data set
matrix	names the matrix containing the data
column_name	names the variables in the data set
row_name	adds a variable containing row titles to the data set

Suppose you want to create a SAS data set named RATIO containing a variable with the height-to-weight ratios for each student, and the ratio is stored in a one column matrix called HTWT. You can use the CREATE and APPEND statements to open a new SAS data set called RATIO and append the observations, naming the data set variable HTWT instead of COL1.

```
CREATE ratio FROM htwt[COLNAME='htwt'] ;
APPEND from htwt;
```

USING THE CREATE STATEMENT WITH THE VAR CLAUSE

You can create a SAS data set from a matrix using the **CREATE** statement with the **VAR** option. This form of the CREATE statement is:

CREATE sas_data_set <VAR operand> ;

where *operand* specifies the variables to be included in the SAS data set.

For example, the following statements create a new SAS data set CLASS having variables NAME, SEX, AGE, HT, and WT. The data come from IML matrices having the same names.

```
CREATE class VAR{name sex age ht wt} ;
APPEND ;
```

You could not do the above using the FROM option because NAME and SEX are character, while AGE, WT, and HT are numeric.

Working with SAS Data Sets: Example 1

```
165  PROC IML;
NOTE: IML Ready
166     USE bios511.class;
167
168     ** list all open data sets **;
169     SHOW DATASETS;
170
171     ** contents of all open data sets **;
172     SHOW CONTENTS;
173
174     ** create column vectors from selected variables **;
175     READ ALL VAR{age ht wt};
176
177     ** list all matrices created so far **;
178     SHOW NAMES;
179
180  QUIT;
NOTE: Exiting IML.
```

```
LIBNAME  MEMNAME                          OPEN MODE   STATUS
-------- -------------------------------- ---------   --------
BIOS511  CLASS                            Input       Current Input

DATASET : BIOS511.CLASS.DATA

 VARIABLE                          TYPE  SIZE
 --------------------------------  ----  ----
 NAME                              char    12
 SEX                               char     1
 AGE                               num      8
 HT                                num      8
 WT                                num      8

Number of Variables   : 5
Number of Observations: 6

 SYMBOL   ROWS   COLS TYPE   SIZE
 ------ ------ ------ ---- ------
 AGE         6      1 num       8
 HT          6      1 num       8
 WT          6      1 num       8
  Number of symbols = 5  (includes those without values)
```

Working With SAS Data Sets: Example 2

```
156  PROC IML;
NOTE: IML Ready
157     USE bios511.class;
158
159     READ ALL VAR _NUM_ INTO x;
160
161     PRINT x;
162
163     CLOSE bios511.class;
164  QUIT;
NOTE: Exiting IML.
NOTE: PROCEDURE IML used:
      real time           0.08 seconds
      cpu time            0.05 seconds
```

```
            X

       37        71       195
       31        70       160
       41        74       195
        .        48         .
        3        12         1
       14        25        45
```

Working With SAS Data Sets: Example 3

```
198  PROC IML;
NOTE: IML Ready
199     USE bios511.class;
200
201     ** create matrix x of males in the class **;
202     READ ALL VAR _NUM_ INTO x WHERE(sex='M');
203
204     ** create matrix containing variable names **;
205     names = {'AGE' 'HT' 'WT'};
206
207     ** print x **;
208     PRINT x[COLNAME=names FORMAT=6.2];
209
210     ** close the data set **;
211     CLOSE bios511.class;
212
213  QUIT;
NOTE: Exiting IML.
NOTE: PROCEDURE IML used:
      real time           0.06 seconds
      cpu time            0.06 seconds
```

```
         X
   AGE     HT     WT

 37.00  71.00 195.00
 31.00  70.00 160.00
 41.00  74.00 195.00
  3.00  12.00   1.00
```

Working With SAS Data Sets: Example 4

```
231  PROC IML;
NOTE: IML Ready
232     USE bios511.class;
233
234     ** create matrix x of males in the class **;
235     READ ALL VAR _NUM_ INTO x[COLNAME=NAMES ROWNAME=name] WHERE(sex='M');
236
237     ** print x **;
238     PRINT 'CLASS Data Set',
239           x[COLNAME=names ROWNAME=name FORMAT=6.2];
240
241     ** close the data set **;
242     CLOSE bios511.class;
243
244  QUIT;
NOTE: Exiting IML.
NOTE: PROCEDURE IML used:
      real time           0.11 seconds
      cpu time            0.06 seconds
```

```
CLASS Data Set
                X
                AGE     HT     WT

CHRISTIANSEN  37.00  71.00 195.00
HOSKING J     31.00  70.00 160.00
HELMS R       41.00  74.00 195.00
FROG K         3.00  12.00   1.00
```

Working With SAS Data Sets: Example 5

```
245  /* Create htwt matrix and then create ratio data set from it */
246  PROC IML;
NOTE: IML Ready
247     USE bios511.class;
248     READ ALL;
249     htwt = ht/wt;
250     SHOW NAMES;
251     CREATE ratio FROM htwt[COLNAME='HTWT'];
252     APPEND FROM htwt;
253     SHOW DATASETS;
254     SHOW CONTENTS;
255     CLOSE ratio;
NOTE: The data set WORK.RATIO has 6 observations and 1 variables.
256  QUIT;
NOTE: Exiting IML.
NOTE: PROCEDURE IML used:
      real time           0.12 seconds
      cpu time            0.06 seconds
```

```
SYMBOL   ROWS   COLS TYPE   SIZE
 ------ ------ ------ ---- ------
 AGE         6      1 num       8
 HT          6      1 num       8
 HTWT        6      1 num       8
 NAME        6      1 char     12
 SEX         6      1 char      1
 WT          6      1 num       8
  Number of symbols = 6  (includes those without values)

LIBNAME  MEMNAME                         OPEN MODE   STATUS
-------- ------------------------------- ---------   --------
BIOS511  CLASS                           Input
WORK     RATIO                           Update      Current Input/Output

DATASET : WORK.RATIO.DATA

 VARIABLE                          TYPE  SIZE
 --------------------------------  ----  ----
 HTWT                              num      8

Number of Variables   : 1
Number of Observations: 6
```

Working With SAS Data Sets: Example 6

```
257  * create matrices from variables, then create a data set from the matrices;
258  PROC IML;
NOTE: IML Ready
259     USE bios511.class;
260     READ ALL VAR{name sex ht wt};
261     SHOW NAMES;
262     CREATE class2 VAR{name sex ht wt};
263     APPEND;
264     SHOW DATASETS;
265     SHOW CONTENTS;
266     CLOSE class2;
NOTE: The data set WORK.CLASS2 has 6 observations and 4 variables.
267  QUIT;
NOTE: Exiting IML.
NOTE: PROCEDURE IML used:
      real time           0.06 seconds
      cpu time            0.06 seconds
```

```
SYMBOL   ROWS   COLS TYPE   SIZE
 ------ ------ ------ ---- ------
 HT          6      1 num       8
 NAME        6      1 char     12
 SEX         6      1 char      1
 WT          6      1 num       8
  Number of symbols = 5  (includes those without values)

LIBNAME  MEMNAME                         OPEN MODE   STATUS
-------- ------------------------------- ---------   --------
BIOS511  CLASS                           Input
WORK     CLASS2                          Update      Current Input/Output

DATASET : WORK.CLASS2.DATA

 VARIABLE                          TYPE  SIZE
 --------------------------------  ----  ----
 NAME                              char    12
 SEX                               char     1
 HT                                num      8
 WT                                num      8

Number of Variables   : 4
Number of Observations: 6
```

OTHER EXAMPLES

A) SOLVING A SYSTEM OF EQUATIONS

$$\begin{aligned} 3X_1 - X_2 + 2X_3 &= 8 \\ 2X_1 - 2X_2 + 3X_3 &= 2 \\ 4X_1 + X_2 - 4X_3 &= 9 \end{aligned}$$

The equations may be written in matrix form as

$$\begin{matrix} 3 & -1 & 2 \\ 2 & -2 & 3 \\ 4 & 1 & -4 \end{matrix} \quad \begin{matrix} X_1 \\ X_2 \\ X_3 \end{matrix} \quad = \quad \begin{matrix} 8 \\ 2 \\ 9 \end{matrix}$$

and can be expressed symbolically as

$AX = C$ or

$X = A^{-1} C$

This system of equations can be solved in IML as follows:

```
 PROC IML;
NOTE: IML Ready
    RESET PRINT;

    a = {3 -1  2,
         2 -2  3,
         4  1 -4};

    c = {8, 2, 9};

    x = INV(a)*c;
 QUIT;
NOTE: Exiting IML.
NOTE: PROCEDURE IML used:
      real time           0.06 seconds
      cpu time            0.06 seconds
```

```
A              3 rows      3 cols    (numeric)

        3        -1         2
        2        -2         3
        4         1        -4


C              3 rows      1 col     (numeric)

        8
        2
        9


X              3 rows      1 col     (numeric)

        3
        5
        2
```

B) LINEAR REGRESSION EXAMPLE

A linear regression model is usually written

Y = XB + E

where Y is a vector of responses, X is the design matrix, and B is a vector of unknown parameters estimated by minimizing the sum of squares of E, the error or residual. The least-squares solution of B is:

$B = (X`X)^{-1}*X`*Y$;

Suppose that you have response data Y measured at 5 values of the independent variables X_1 and X_2 and you want to solve the following equation for the unknown parameters B_1 and B_2

$Y = B_1X_1 + B_2X_2$

```
PROC IML;
NOTE: IML Ready
   RESET PRINT;

   x = {1 1 1,
        1 2 4,
        1 3 9,
        1 4 16,
        1 5 25} ;

   y = {1, 5, 9, 23, 36} ;

   b = INV(x`*x)*x`*y ;

QUIT;
NOTE: Exiting IML.
```

```
X               5 rows      3 cols    (numeric)

         1         1         1
         1         2         4
         1         3         9
         1         4        16
         1         5        25


Y               5 rows      1 col     (numeric)

         1
         5
         9
        23
        36


B               3 rows      1 col     (numeric)

       2.4
      -3.2
         2
```