SAS/IML - Interactive Matrix Language

- SAS/IML 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 are 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

3476    PROC IML;
IML Ready
3477    RESET PRINT;
3478
3479    A=3;
3480
3481    B={1 2 3};
3482
3483    C={1, 2, 3};
3484
3485    D={1 2 3, 4 5 6};
3486
3487    QUIT;
Exiting IML.
NOTE: The PROCEDURE IML used 0.22 seconds.

<table>
<thead>
<tr>
<th></th>
<th>row</th>
<th>col</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
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
- 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 have many elements arranged in rectangular form.
- if there are multiple elements, use braces( { } ) to enclose the values and commas to separate the rows.
- all rows must have the same number of elements.

C) Scaler Literals
a=12 ;
a=. ;
a='bios 111'
a="BIOS111" ;

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 you singe 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 create a 2X3 numeric matrix named A .

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

The statement

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

redefines matrix A as a 1X3 character matrix.
Scaler Literals Example

84 PROC IML;
85 RESET PRINT;
86
87 A = 12;
88 B = .;
89 C = 'bios 111';
90 d = "BIOS111";
91
92 QUIT;
Exiting IML.
NOTE: The PROCEDURE IML used 0.71 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 111

D 1 row 1 col (character, size 7)
BIOS111
Numeric Literals Example

93 TITLE "NUMERIC LITERALS" ;
94 PROC IML ;
 IML Ready
95   RESET PRINT ;
96
97   X = {1 2 3 4 5 6} ;
98   Y = {1,2,3,4,5} ;
99   Z = {1 2, 3 4, 5 6} ;
100
101 QUIT ;
 Exiting IML.
NOTE: The PROCEDURE IML used 4.0 seconds.

NUMERIC LITERALS

X             1 row       6 cols    (numeric)
     1       2       3       4      5        6

Y             5 rows      1 col     (numeric)
1
2
3
4
5

Z             3 rows      2 cols    (numeric)
     1       2
     3       4
     5       6
Character Literals Example

93  TITLE "CHARACTER LITERALS" ;
94  PROC IML ;
IML Ready
95  RESET PRINT ;
96
97  A = {abc defg} ;
98  B = {'abc' 'defg'} ;
99  B = {1 2 3, 4 5 6} ;
100
101  QUIT ;
Exiting IML.

NOTE: The PROCEDURE IML used 4.0 seconds.

CHARACTER LITERALS

A             1 row       2 cols    (character, size)

   ABC   DEFG

B             1 row       2 cols    (character, size)

   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 a new matrix.

OPERATORS

There are 3 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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
<th>Syntax type</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign reverse</td>
<td>-</td>
<td>Prefix</td>
<td>Num</td>
</tr>
<tr>
<td>Addition</td>
<td>+</td>
<td>Infix</td>
<td>Num</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>Infix</td>
<td>Num</td>
</tr>
<tr>
<td>Index creation</td>
<td>:</td>
<td>Infix</td>
<td>Num</td>
</tr>
<tr>
<td>Matrix multiplication</td>
<td>*</td>
<td>Infix</td>
<td>Num</td>
</tr>
<tr>
<td>Elementwise multiplication</td>
<td>#</td>
<td>Infix</td>
<td>Num</td>
</tr>
<tr>
<td>Matrix power</td>
<td>**</td>
<td>Infix</td>
<td>Num</td>
</tr>
<tr>
<td>Elementwise power</td>
<td>##</td>
<td>Infix</td>
<td>Num</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>Infix</td>
<td>Num</td>
</tr>
<tr>
<td>Horizontal concatenation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vertical concatenation</td>
<td>//</td>
<td>Infix</td>
<td>Both</td>
</tr>
<tr>
<td>Element maximum</td>
<td>&lt;&gt;</td>
<td>Infix</td>
<td>Both</td>
</tr>
<tr>
<td>Element minimum</td>
<td>&gt;&lt;</td>
<td>Infix</td>
<td>Both</td>
</tr>
<tr>
<td>Logical AND</td>
<td>&amp;</td>
<td>Infix</td>
<td>Num</td>
</tr>
<tr>
<td>Logical OR</td>
<td></td>
<td></td>
<td>Infix</td>
</tr>
<tr>
<td>Logical NOT</td>
<td>^</td>
<td>Prefix</td>
<td>Num</td>
</tr>
<tr>
<td>Less than</td>
<td>&lt;</td>
<td>Infix</td>
<td>Both</td>
</tr>
<tr>
<td>Greater than</td>
<td>&gt;</td>
<td>Infix</td>
<td>Both</td>
</tr>
<tr>
<td>Equal to</td>
<td>=</td>
<td>Infix</td>
<td>Both</td>
</tr>
<tr>
<td>Less than or equal to</td>
<td>&lt;=</td>
<td>Infix</td>
<td>Both</td>
</tr>
<tr>
<td>Greater than or equal to</td>
<td>&gt;=</td>
<td>Infix</td>
<td>Both</td>
</tr>
<tr>
<td>Not equal to</td>
<td>^=</td>
<td>Infix</td>
<td>Both</td>
</tr>
<tr>
<td>Transpose</td>
<td>\</td>
<td>Postfix</td>
<td>Both</td>
</tr>
<tr>
<td>Subscripts</td>
<td>[ ]</td>
<td>Postfix</td>
<td>Both</td>
</tr>
</tbody>
</table>

Table A1.2  Subscript Reduction Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>#</td>
<td>Multiplication</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Maximum</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>Minimum</td>
</tr>
<tr>
<td>&lt;:::</td>
<td>Index of maximum</td>
</tr>
<tr>
<td>::&lt;</td>
<td>Index of minimum</td>
</tr>
<tr>
<td>:</td>
<td>Mean</td>
</tr>
<tr>
<td>##</td>
<td>Sum of squares</td>
</tr>
</tbody>
</table>
**EXAMPLES:**

\[
\begin{array}{ccc}
\text{LET} & \text{A} & 2 & 2 & \text{and} & \text{B} & 4 & 5 \\
 & 3 & 4 & & & & 1 & 0 \\
\end{array}
\]

A + B yields 
\[
\begin{array}{cc}
6 & 7 \\
4 & 4 \\
\end{array}
\]
addition

A # B yields 
\[
\begin{array}{cc}
8 & 10 \\
3 & 0 \\
\end{array}
\]
elementwise

A * B yields 
\[
\begin{array}{cc}
10 & 10 \\
16 & 15 \\
\end{array}
\]
matrix

A ##2 yields 
\[
\begin{array}{cc}
4 & 4 \\
9 & 16 \\
\end{array}
\]
element power

A <> B yields 
\[
\begin{array}{cc}
4 & 5 \\
3 & 4 \\
\end{array}
\]
operator

A <= B yields 
\[
\begin{array}{cc}
1 & 1 \\
0 & 0 \\
\end{array}
\]
less than or equal to operator

A\' yields 
\[
\begin{array}{cc}
2 & 3 \\
2 & 4 \\
\end{array}
\]
transpose operator

A || B yields 
\[
\begin{array}{cccc}
2 & 2 & 4 & 5 \\
3 & 4 & 1 & 0 \\
\end{array}
\]
horizontal

A // B yields 
\[
\begin{array}{cccc}
2 & 2 & 3 & 4 \\
4 & 5 & 1 & 0 \\
\end{array}
\]
vertical
Operators Example

123 PROC IML;
124   IML Ready
125
126   A = {2 2, 3 4} ;
127   B = {4 5, 1 0} ;
128   C = -B ;
129   D = A + B ;
130   E = A' ;
131
132 QUIT ;
Exiting IML.
NOTE: The PROCEDURE IML used 0.5 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
IML: SUBSCRIPTS

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

\[ \text{operand}[\text{row},\text{column}] \]

where

\( \text{operand} \) is usually a matrix name

\( \text{row} \) is an expression selecting 1 or more rows

\( \text{column} \) is an expression selecting 1 or more columns

**subscripts are used to:**

- refer to a single element of a matrix

- refer to 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 1 subscript to look for the element down the rows.

For example:

\[
\begin{array}{ccc}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{array}
\]

\[B=A[2,3] \quad \text{yields} \quad 6\]

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


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 \(A\).

\[C=A[1,] ;\]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
\end{array}
\]

To select column 2 of \(A\).

\[D=A[,]2] ;\]

\[
\begin{array}{c}
2 \\
5 \\
8 \\
\end{array}
\]
SUBMATRICES

You can refer to 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:

\[ E = A[\{1,3\},\{1,2\}] \]

\[
\begin{bmatrix}
1 & 2 \\
7 & 8
\end{bmatrix}
\]

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:

\[ \text{LET } B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix} \]

\[ F = B[1:3,1:2] \]

\[
\begin{bmatrix}
1 & 2 \\
5 & 6 \\
9 & 10
\end{bmatrix}
\]

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:

\[ A[,3]=[20 30 40] \]
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 (sum across the rows, which reduces the row dimension to 1), specify X[+,1]. 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 lists the eight operators for subscript reduction.

Subscript Reduction Operators Examples

PROC IML;
    RESET PRINT ;
    A={0 1 2,5 4 3,7 6 8} ;
    B=A[,+] ;
    C=A[:.,] ;
    D=A[+,] ;
    E=A[:,] ;
    QUIT;

A             3 rows      3 cols    (numeric)
    0         1         2
    5         4         3
    7         6         8

B             3 rows      1 col     (numeric)
    3
    12
    21
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:

\[
\text{result=FUNCTION(arguements)} \; ;
\]

For example:

\[
A=\text{SQRT}(B) \; ;
\]

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

An example of how to use the ABS function is

    C=ABS(A) ;

THE SQRT FUNCTION

SQRT(matrix) ;

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

An example of how to use the SQRT function is

    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

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 logarithm of every element in the input matrix, where matrix is a numeric matrix or literal

An example of how to use the LOG function is

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

\[ C = \text{MAX}(A) ; \]

Note: there is also a MIN function that works the same way.

**THE DET FUNCTION**

\[ \text{DET}(\text{matrix}) ; \]

Returns a scalar containing the determinant of a square matrix, where \( \text{matrix} \) is a numeric matrix or literal

An example of how to use the DET function is

\[ C = \text{DET}(A) ; \]

**THE INV FUNCTION**

\[ \text{INV}(\text{matrix}) ; \]

Returns the inverse of a matrix, where \( \text{matrix} \) is a square and nonsingular matrix.

An example of how to use the INV function is

\[ C = \text{INV}(A) ; \]

**THE DIAG FUNCTION**

\[ \text{DIAG}(\text{matrix}) ; \]

where \( \text{matrix} \) 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,

\[ A = \begin{bmatrix} 4 & 3 \\ 2 & 1 \end{bmatrix} ; \]

\[ C = \text{DIAG}(A) \; \text{results in} \]

\[ C \begin{array}{ll} 2 \text{ rows} & 2 \text{ cols} \\ \hline 4 & 0 \\ 0 & 1 \end{array} \] (numeric)

\[ B = \{1 \ 2 \ 3\} ; \]
\[ D = \text{DIAG}(B) \; \text{results in} \]

\[ D \begin{array}{llll} 3 \text{ rows} & 3 \text{ cols} \\ \hline 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{array} \] (numeric)

**THE VECDIAG FUNCTION**

\[ \text{VECDIAG}(\text{matrix}) ; \]

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

\[ A = \begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix} ; \]

\[ C = \text{DIAG}(A) ; \] results in

\[ C \begin{array}{ccc}
\text{2 rows} & \text{1 cols} & \text{(numeric)} \\
4 & \\
1
\end{array} \]
Functions Example

proc logistic data=in.v2 covout
  outest=out(drop=_link_ _type_ _name_ _lnlike_) noprint ;
  model smoker = sysba chol trig alcohol;
run;

PROC IML;
RESET PRINT ;
USE OUT;
READ ALL INTO VB ; /* betas/variance covariance matrix */
NVB= NROW(VB);     /* number of rows in VB */
BETA=VB[1,];       /* row vector or 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 = BETA || L || H ; /* lower CI, betas, upper CI */
QUIT;

OUT DATA SET

<table>
<thead>
<tr>
<th>OBS</th>
<th>INTERCEPT</th>
<th>SYSBA</th>
<th>CHOL</th>
<th>TRIG</th>
<th>ALCOHOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.10335</td>
<td>0.01019</td>
<td>-0.01004</td>
<td>.00023</td>
<td>-0.02835</td>
</tr>
<tr>
<td>2</td>
<td>2.94187</td>
<td>-0.01546</td>
<td>-0.00389</td>
<td>.00001</td>
<td>-0.00135</td>
</tr>
<tr>
<td>3</td>
<td>-0.01546</td>
<td>0.00014</td>
<td>-0.00001</td>
<td>.00000</td>
<td>-0.00002</td>
</tr>
<tr>
<td>4</td>
<td>-0.00389</td>
<td>-0.00001</td>
<td>0.00002</td>
<td>.00000</td>
<td>0.00001</td>
</tr>
<tr>
<td>5</td>
<td>0.00001</td>
<td>0.00000</td>
<td>0.00000</td>
<td>.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>6</td>
<td>-0.00135</td>
<td>-0.00002</td>
<td>0.00001</td>
<td>.00000</td>
<td>0.00016</td>
</tr>
</tbody>
</table>
**IML Output**

**VB**
6 rows 5 cols (numeric)

| 2.10335 | 0.01019 | -0.01004 | 0.00023 | -0.02835 |
| 2.94187 | -0.01546 | -0.00389 | 0.00001 | -0.00135 |
| -0.01546 | 0.00014 | -0.00001 | 0 | -0.00002 |
| -0.00389 | -0.00001 | 0.00002 | 0 | 0.00001 |
| 0.00001 | 0 | 0 | 0 | 0 |
| -0.00135 | -0.00002 | 0.00001 | 0 | 0.00016 |

**NVB**
1 row 1 col (numeric)

6

**BETA**
1 row 5 cols (numeric)

| 2.10335 |
| 0.01019 |
| -0.01004 |
| 0.00023 |
| -0.02835 |

**BETA**
5 rows 1 col (numeric)

| 2.10335 |
| 0.01019 |
| -0.01004 |
| 0.00023 |
| -0.02835 |

**V**
5 rows 5 cols (numeric)

<p>| 2.94187 | -0.01546 | -0.00389 | 0.00001 | -0.00135 |
| -0.01546 | 0.00014 | -0.00001 | 0 | -0.00002 |
| -0.00389 | -0.00001 | 0.00002 | 0 | 0.00001 |
| 0.00001 | 0 | 0 | 0 | 0 |
| -0.00135 | -0.00002 | 0.00001 | 0 | 0.00016 |</p>
<table>
<thead>
<tr>
<th>Column</th>
<th>Rows</th>
<th>Columns</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>5</td>
<td>1</td>
<td>numeric</td>
</tr>
<tr>
<td></td>
<td>2.94187</td>
<td>0.00014</td>
<td>0.00002</td>
</tr>
<tr>
<td>S</td>
<td>5</td>
<td>1</td>
<td>numeric</td>
</tr>
<tr>
<td></td>
<td>1.715188</td>
<td>0.0118322</td>
<td>0.0044721</td>
</tr>
<tr>
<td>L</td>
<td>5</td>
<td>1</td>
<td>numeric</td>
</tr>
<tr>
<td></td>
<td>-1.258419</td>
<td>-0.013001</td>
<td>-0.018805</td>
</tr>
<tr>
<td>H</td>
<td>5</td>
<td>1</td>
<td>numeric</td>
</tr>
<tr>
<td></td>
<td>5.4651186</td>
<td>0.033381</td>
<td>-0.001275</td>
</tr>
<tr>
<td>LBH</td>
<td>5</td>
<td>3</td>
<td>numeric</td>
</tr>
<tr>
<td></td>
<td>2.10335</td>
<td>-1.258419</td>
<td>5.4651186</td>
</tr>
</tbody>
</table>
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 DATASET

- 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 dataset, submit a USE command to open it. The general form of the USE statement is:
  
  USE sas_dataset <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_dataset <VAR operand> <WHERE expression>;

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

  CREATE sas_dataset <VAR operand>;
  CREATE sas_dataset FROM from_name
  <[COLNAME=column_name ROWNAME=row_name]> ;

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

READING OBSERVATION FROM A SAS DATASET

- 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 classlib.class ;
READ all VAR{age height weight} ;
```

The read statement above creates 3 numeric vectors AGE, HEIGHT, and WEIGHT. The two statements above can also be written as:

```
USE classlib 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 classlib.class ;
READ all VAR _num_ INTO X ;
```

USING THE READ STATEMENT WITH THE WHERE CLAUSE

You can use the 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 classlib.class ;
READ all VAR _num_ INTO female WHERE(sex="F");
```

CREATING A SAS DATASET 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_dataset FROM matrix
    <[COLNAME=column_name ROWNAME=row_name]> ;
```

where

- **sas_dataset** 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 tiles 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 dataset from a matrix using the **CREATE** statement with the **VAR** option. This form of the CREATE statement is:

```
CREATE sas_dataset <VAR operand> ;
```

where operand are 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 name.

```
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: Examples

20   PROC IML ;
     IML Ready
21   USE SC.CLASS ;
23   ** LIST ALL OPEN DATASETS ** ;
25   SHOW DATASETS ;
27   ** CONTENTS OF ALL OPEN DATASETS ** ;
29   SHOW CONTENTS ;
31   READ ALL VAR{AGE HT WT} ;
33   ** LIST ALL MATRICES CREATED SO FAR ** ;
35   SHOW NAMES ;
36
37   QUIT ;
     Exiting IML.
     NOTE: The PROCEDURE IML used 0.32 seconds.

     LIBNAME MEMNAME    OPEN MODE   STATUS
                 ------- -------   ---------   ------
SC      .CLASS    Input       Current Input

     DATASET : SC.CLASS.DATA

     VAR NAME   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

     AGE           6 rows      1 col  num      8
     HT            6 rows      1 col  num      8
     WT            6 rows      1 col  num      8
     Number of symbols = 5  (includes those without values)
### Working With SAS Datasets: Examples

```sas
PROC IML;
IML Ready
USE SC.CLASS;
READ ALL VAR _NUM_ INTO X;
** PRINT X **;
PRINT X;
** CLOSE SC.CLASS **;
CLOSE SC.CLASS;
QUIT;
```

Exiting IML.

NOTE: The PROCEDURE IML used 4.78 seconds.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>71</td>
<td>195</td>
</tr>
<tr>
<td>31</td>
<td>70</td>
<td>160</td>
</tr>
<tr>
<td>41</td>
<td>74</td>
<td>195</td>
</tr>
<tr>
<td>.</td>
<td>48</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>25</td>
<td>45</td>
</tr>
</tbody>
</table>
Working With SAS Data Sets: Examples

69  PROC IML;
    IML Ready
70  USE SC.CLASS;
71
72  READ ALL VAR _NUM_ INTO X WHERE(SEX='M');
73
74  ** CREATE MATRIX CONTAINING VARIABLE **
    **                     NAMES **;
75
76  NAMES={"AGE" "HT" "WT"};
77
78  ** PRINT X **;
79
80  PRINT X[COLNAME=NAMES FORMAT=6.2];
81
82  ** CLOSE SC.CLASS **;
83
84  CLOSE SC.CLASS;
85  QUIT;
Exiting IML.
NOTE: The PROCEDURE IML used 2.37 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
PROC IML;
IML Ready
USE SC.CLASS;
READ ALL VAR _NUM_
    INTO X[COLNAME=NAMES ROWNAME=NAME]
    WHERE(SEX='M');
** PRINT X **;
PRINT "CLASS DATESET",
    X[COLNAME=NAMES ROWNAME=NAME FORMAT=6.2];
** CLOSE SC.CLASS **;
CLOSE SC.CLASS;
QUIT;
Exiting IML.
NOTE: The PROCEDURE IML used 0.28 seconds.

_________________________________________
CLASS DATESET

<table>
<thead>
<tr>
<th>X</th>
<th>AGE</th>
<th>HT</th>
<th>WT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHRISTIANSEN</td>
<td>37.00</td>
<td>71.00</td>
<td>195.00</td>
</tr>
<tr>
<td>HOSKING J</td>
<td>31.00</td>
<td>70.00</td>
<td>160.00</td>
</tr>
<tr>
<td>HELMS R</td>
<td>41.00</td>
<td>74.00</td>
<td>195.00</td>
</tr>
<tr>
<td>FROG K</td>
<td>3.00</td>
<td>12.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Working With SAS Datasets: Examples

36   PROC IML;
37     USE SC.CLASS;
38     READ ALL;
39     HTWT=HT/WT;
40     SHOW NAMES;
41     CREATE RATIO FROM HTWT[COLNAME='HTWT'];
42     APPEND FROM HTWT;
43     SHOW DATASETS;
44     SHOW CONTENTS;
45     CLOSE RATIO;

NOTE: The data set WORK.RATIO has 6 observations and 1 variables.

46    QUIT;
Exiting IML.
NOTE: The PROCEDURE IML used 0.66 seconds.

<table>
<thead>
<tr>
<th></th>
<th>AGE</th>
<th>6 rows</th>
<th>1 col</th>
<th>num</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HT</td>
<td>6 rows</td>
<td>1 col</td>
<td>num</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>HTWT</td>
<td>6 rows</td>
<td>1 col</td>
<td>num</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
<td>6 rows</td>
<td>1 col</td>
<td>char</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>SEX</td>
<td>6 rows</td>
<td>1 col</td>
<td>char</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>WT</td>
<td>6 rows</td>
<td>1 col</td>
<td>num</td>
<td>8</td>
</tr>
</tbody>
</table>

LIBNAME MEMNAME   OPEN MODE   STATUS
------- -------   ---------   -----
SC      .CLASS    Input
WORK    .RATIO    Update      Current Input      Current Output

DATASET : WORK.RATIO.DATAT

<table>
<thead>
<tr>
<th>VAR NAME</th>
<th>TYPE</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTWT</td>
<td>NUM</td>
<td>8</td>
</tr>
</tbody>
</table>

Number of Variables: 1
Number of Observations: 6
Working With SAS Datasets: Examples

47  PROC IML;
48    USE SC.CLASS;
49    READ ALL VAR{NAME SEX HT WT};
50    SHOW NAMES;
51    CREATE RATIO2 VAR{NAME SEX HT WT};
52    APPEND;
53    SHOW DATASETS;
54    SHOW CONTENTS;
55    CLOSE RATIO2;
NOTE: The data set WORK.RATIO2 has 6 observations and 4 variables.
56    QUIT;
Exiting IML.
NOTE: The PROCEDURE IML used 7.96 seconds.

_________________________________________
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HT</td>
<td>6</td>
<td>rows</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
<td>6</td>
<td>rows</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SEX</td>
<td>6</td>
<td>rows</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>WT</td>
<td>6</td>
<td>rows</td>
<td>1</td>
</tr>
</tbody>
</table>

LIBNAME MEMNAME OPEN MODE STATUS
------- ------- --------- ------
SC       .CLASS Input
WORK     .RATIO2 Update Current Input Current Output

DATASET : WORK.RATIO2.DATA

<table>
<thead>
<tr>
<th>VAR</th>
<th>NAME</th>
<th>TYPE</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NAME</td>
<td>CHAR</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>SEX</td>
<td>CHAR</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>HT</td>
<td>NUM</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>WT</td>
<td>NUM</td>
<td>8</td>
</tr>
</tbody>
</table>
Number of Variables: 4
Number of Observations: 6
A) SOLVING A SYSTEM OF EQUATIONS

\[ \begin{align*}
3X_1 - 2X_2 + 2X_3 &= 8 \\
2X_1 - 2X_2 + 3X_3 &= 2 \\
4X_1 + X_2 - 4X_3 &= 9 
\end{align*} \]

The equations may be written in matrix form as

\[
\begin{bmatrix}
3 & -1 & 2 \\
2 & -2 & 3 \\
4 & 1 & -4 \\
\end{bmatrix}
\begin{bmatrix}
X_1 \\
X_2 \\
X_3 \\
\end{bmatrix}
=
\begin{bmatrix}
8 \\
2 \\
9 \\
\end{bmatrix}
\]

and can be expressed symbolically as

\[
AX = C \quad \text{or} \quad X = A^{-1} C
\]

This system of equations can be solved in IML using the IML statements:

```plaintext
5    PROC IML;
IML Ready
6     RESET PRINT ;
7     A={3 -1  2,
8           2 -2  3,
9           4  1 -4} ;
10
11    C={8, 2, 9} ;
12
13    X=INV(A)*C ;
14   QUIT ;
Exiting IML.
```

NOTE: The PROCEDURE IML used 4.67 seconds.
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 \( B_2 \)

\[ Y = B_1X_1 + B_2X_2 \]
28 PROC IML;
29   RESET PRINT ;
30   X={1 1 1,
31       1 2 4,
32       1 3 9,
33       1 4 16,
34       1 5 25} ;
35   Y={1, 5, 9, 23, 36} ;
36   B=INV(X`*X)*X`*Y ;
37 QUIT ;
Exiting IML.

<table>
<thead>
<tr>
<th>X</th>
<th>5 rows</th>
<th>3 cols</th>
<th>(numeric)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>25</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Y</th>
<th>5 rows</th>
<th>1 col</th>
<th>(numeric)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B</th>
<th>3 rows</th>
<th>1 col</th>
<th>(numeric)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-3.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>