Chapter 4

Arrays

In MATLAB, a matrix (a 2-dimensional array) is a rectangular array of numbers. Special meaning is sometimes attached to 1-by-1 matrices, which are scalars, and to matrices with only one row or column, which are vectors. MATLAB has other ways of storing both numeric and non-numeric data, but in the beginning, it is usually best to think of everything as a matrix. The operations in MATLAB are designed to be as natural as possible. Where other programming languages work with numbers one at a time, MATLAB allows you to work with entire matices quickly and easily.

A matrix (or array) of order **m** by **n** is simply a set of numbers arranged in a rectangular block of m horizontal rows and n vertical columns. The following

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}$$

is a matrix of size (m by n). Sometimes we say “matrix A has dimension (m by n).” The numbers that make up the array are called the elements of the matrix an, in MATLAB, no distinction is made between elements that are real numbers and complex numbers. In the double subscript notation a_{ij} for matrix element a(i,j), the first subscript i denotes the row number, and the second subscript j denotes the column numbers.

Note: 1) All variables in MATLAB are arrays. A scalar is an array with one element; a vector is an array with one row or one column; a matrix is an array with multiple rows and columns
2) The variable (scalar, vector, or array) is defined by the input when the variable is initialized (assigned value). There is no need to define the size of the array before the elements are assigned.
3) Once a variable exists (a scalar, vector, matrix), it can be changed to be any other size, or type, or variable.

4.1 Creating a One-Dimensional Array (vector)

A one-dimensional array is a list of numbers that is placed into a row or a column. Any list of numbers can be set up as a vector. A row vector is simply a (1 by n) matrix and a column vector is a (m by 1) matrix. The ith element of a vector

$$V = [v_1 \quad v_2 \quad v_3 \quad v_4 \quad \dots \quad v_n]$$

is simply denoted v_i. The MATLAB language has been designed to make the definition and manipulation of matrices and vectors as simple as possible.

Row vector: To create a row vector type the elements with a space or a comma between the elements inside the square brackets.

```
>> dates = [1  4  10  17  25]

dates =

     1     4    10    17    25

or

yr = [1984, 1986, 1988, 1990, 1992, 1994, 1996]

yr =

        1984        1986        1988        1990        1992        1994        1996
```

Column vector: To create a row vector type the elements with a semicolon between the elements inside the square brackets.

```
pop = [127; 130; 136; 145; 158; 178; 211]

pop =

   127
   130
   136
   145
   158
   178
   211
```

A second way to create a column vector is to use an **Enter** to indicate the next element.

```
>> mom = [11
13
21
45
62]

mom =

    11
    13
    21
    45
    62
```

4.2 Colon Notation in Creating Vectors

Create a vector with constant spacing by specifying the first term, the spacing, and the last term:

In a vector with constant spacing the difference between the elements is the same. For example, in the vector: `v = 2  4  6  8  10`, the spacing between the elements is 2. A vector in which the first term is `m`, the spacing is `q`, and the last term is `n` is created by typing:

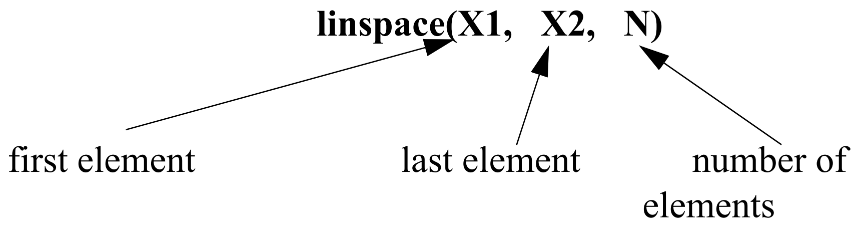
```
variable_name = [m:q:n] or
variable_name = m:q:n       % brackets are optional

Example 1:
>> x = [1:2:13]                        first element:  1
                                       spacing: 2
x =                                    last element: 13
     1     3     5     7     9    11    13

Example 2:
>> y=[1.5:0.1:2.1]                     first element:  1.5
                                       spacing:  0.1
y =                                    last element:  2.1
    1.5000    1.6000    1.7000    1.8000    1.9000    2.0000    2.1000


Example 3:
>> z=[-3:7]                             first element:  -3
                                        spacing (if omitted):  1
z =                                     last element:  7
    -3    -2    -1     0     1     2     3     4     5     6     7


Example 4:
>> q = [21:-3:6]                        first element:  21
                                        spacing:  -3
q =                                     last element:  6

    21    18    15    12     9     6
```

4.3 Linear Spacing: linspace

A vector in which the first element is X1, the last element is X2, and the number of elements is N is created by typing the **linspace** command (MATLAB determines the correct spacing):

:

General Format:



If N is omitted, linspace(X1, X2) generates a row vector of 100 linearly equally spaced points between X1 and X2

For N < 2, linspace returns X2.

4.3.1 Examples:

```
>> x = linspace(0,8,6)                #elements: 6
                                      first element:  0
x =                                   last element:  8

         0    1.6000    3.2000    4.8000    6.4000    8.0000


>> y = linspace(30,10,11)            #elements:  11
                                     first element: 30
y =                                  last element:  10

    30    28    26    24    22    20    18    16    14    12    10



>> z = linspace(49,0.5)              #elements:  100  (by default
                                                      #elements = 100)
z =                                  first element:  49
                                     last element:  0.5
  Columns 1 through 10

49.0000  48.5101  48.0202  47.5303  47.0404  46.5505  46.0606  45.5707  45.0808
44.5909

........
  Columns 90 through 100

5.3990  4.9091  4.4192  3.9293  3.4394  2.9495  2.4596  1.9697  1.4798  0.9899
0.5000
```

4.4 Creating Two-Dimensional Array (Matrix)

4.4.1 By Enumeration

In a square matrix, the number of rows and number of columns are equal.

```
    1   2   3
    4   5   6                    a   3  by  3  square matrix
    7   8   9
```

In general, the number of rows and columns may be different.

```
  1  2  3  4  5  6
  7  8  9  1  2  3               a  3  by  6  matrix
  4  5  6  7  8  9
has 3 rows and 6 columns.
```

A matrix is created by assigning the elements of the matrix to a variable. This is done by typing the elements, row by row, inside square brackets [ ]. First type the left bracket [, then type the first row separating the elements with spaces or commas. To type the next row type a semicolon or press **Enter**. Type the right bracket] at the end of the last row.

variable_name = [1st row elements; 2nd row elements; 3rd row elements; ... ; last row elements]

The elements that are entered can be numbers or mathematical expressions, predefined variables, and functions. All the rows MUST have the same number of elements. If an element is zero, it has to be entered as such. MATLAB displays an error message if an attempt is made to define an incomplete matrix.

```
>> X=[1 5;2 1]

X =

     1     5
     2     1


>> Y = [7   2  76  33  2
        1   9   5   3  2
        5  18  22  32  6]

Y =

     7     2    76    33     2
     1     9     5     3     2
     5    18    22    32     6


>> A = [1:2:11; 0:5:25; linspace(10,60,6);5 25 30 35 40 45]

A =

     1     3     5     7     9    11
     0     5    10    15    20    25
    10    20    30    40    50    60
     5    25    30    35    40    45
```

4.5 Indexing of Arrays

`x(i)` refers to the `ith` element of array `x`

There is no off-by-one issue!!!

```
>> x  = [1 3 5 7];
x(1) is the first element of array x, i.e., x(1) = 1
x(4) is the 4th element of array x, i.e.,   x(4) = 7

>> y = [2 4; 5 7]

y =

     2     4
     5     7

y(1,1) = 2    row 1  column 1
y(2,1) = 5    row 2  column 1
y(1,2) = 4    row 1  column 2
y(2,2) = 7    row 2  column 2
```

4.6 Arrays Automatically Resize

```
>> A = [6 9 4;
        1 5 7];

A(1,5) = 3

A =

     6     9     4     0     3
     1     5     7     0     0
```

4.7 Specialty Matrices

MATLAB provides multiple functions that generate basic matrices.

`zeros(r,c)`	All zeros
`ones(r,c)`	All ones
`eye(r,r)`	Ones down the main diagonal
`rand(r,c)`	Uniformly distributed random elements
`randn(r,c)`	Normally distributed random elements
`magic(n)`	Creates magic squares of size n
`Z = []`	Empty array

4.7.1 Examples Specialty Matrices:

zeros

```
>> Z = zeros(2,4)

Z =

     0     0     0     0
     0     0     0     0
```

ones

```
>> F = 5 * ones(3,3)

F =

     5     5     5
     5     5     5
     5     5     5
```

eye

```
>> E = eye(3,3)

E =

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

rand

```
>> N = fix(10*rand(1,10))

N =

     9     2     6     4     8     7     4     0     8     4
```

randn

```
>> R = randn(4,4)

R =

   -0.4326   -1.1465    0.3273   -0.5883
   -1.6656    1.1909    0.1746    2.1832
    0.1253    1.1892   -0.1867   -0.1364
    0.2877   -0.0376    0.7258    0.1139
```

magic

```
>> M = magic(4)

M =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

4.8 load Command

The load command reads binary files containing matrices generated by earlier MATLAB sessions, or reads text files containing numeric data. The text file should be organized as a rectangular table of numbers, separated by blanks, with one row per line, and an equal number of elements in each row. For example, outside of MATLAB< create a text file containing these four lines:

```
  16.0    3.0    2.0    13.0
   5.0   10.0   11.0     8.0
   9.0    6.0    7.0    12.0
   4.0   15.0   14.0     1.0
```

Store the file under the name of `ML.dat`. Then the command

```
   load  ML.dat
```

reads the file and creates a variable, `ML`, containing our example matrix.

An easy way to read data into MATLAB in many text or binary formats is to use the Import Wizard.

4.9 Concatenation

Concatenation is the process of joining small matrices to make bigger ones. In fact, you made your first matrix by concatenating its individual elements. The pair of square brackets, [ ] , is the concatenation operator. For an example, start with the 4- by 4 magic square, M, and form

```
B = [M   M+32 ;  M+48  M+16]
```

```
>> M = magic(4)

M =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1



>> B = [M   M+32 ;  M+48  M+16]

B =

    16     2     3    13    48    34    35    45
     5    11    10     8    37    43    42    40
     9     7     6    12    41    39    38    44
     4    14    15     1    36    46    47    33
    64    50    51    61    32    18    19    29
    53    59    58    56    21    27    26    24
    57    55    54    60    25    23    22    28
    52    62    63    49    20    30    31    17
```

```
r = [2, 4, 20];
w = [9, -6, 3];
u = [r, w] gives
    [ 2, 4, 20, 9, -6, 3]

A = [1 2 3; 4 5 6; 7 8 9]
B = [A, zeros(3,2); zeros(2,3), eye(2)]

B =

     1     2     3     0     0
     4     5     6     0     0
     7     8     9     0     0
     0     0     0     1     0
     0     0     0     0     1
```

4.10 SubVectors

```
v( : ) refers to all elements of v

v( m : n ) refers to elements m through n

Example 1:
>> V = [1 2 3 4]

V =

     1     2     3     4

>> V(:)

ans =

     1
     2
     3
     4


Example 2:
>> V(2:4)

ans =

     2     3     4
```

4.11 Submatrices

Any matrix obtained by omitting some rows and columns from a given matrix X is called a "sub-matrix" of X. The colon notation can be used to pick out selected row, columns, and elements of vectors, matrices, and arrays. The colon may be viewed as a "wild-card" character.

X(:)   is all the elements of X, regarded as a single column
X(:, n) refers to the elements in column n (all rows)
X(n , :) refers to the elements in row n (all columns)
X(: , m : n) refers to all elements in all rows between columns m and n
X(m : n , p : q) refers to all elements in rows m through n and columns p through q

4.11.1 Examples:

```
Example 1:
X =

     1     2     3
     4     5     6

>> X(:)

ans =

     1
     4
     2
     5
     3
     6
```

```
Example 2:
   >> A(1:4, 3)
```

is the column vector consisting of the first four entries of the third column of A. A colon by itself denotes an entire row or column.

```
A =

     1     2     3     4     5
     6     7     8     9    10
    11    12    13    14    15
    16    17    18    19    20
    21    22    23     2    25
    26    27    28    29    30

>> A(1:4, 3)

ans =

     3
     8
    13
    18
```

Example 3:

```
   >> A(:, 3)
```
is the third column of A, and A(1 : 4 , :) is the first four rows of A. Arbitrary integral vectors can be used as subscripts.
```
>> A(:,3)

ans =

     3
     8
    13
    18
    23
    28

>> A(1:4,:)

ans =

     1     2     3     4     5
     6     7     8     9    10
    11    12    13    14    15
    16    17    18    19    20
```

Example 4:
```
   >> A(:, [2  4])
```
generates a two-column matrix containing columns 2 and 4 of matrix A. This subscripting scheme can be used on both sides of an assignment statement.
```
>> A(:, [2 4])

ans =

     2     4
     7     9
    12    14
    17    19
    22     2
    27    29
```

Example 5:
```
   >> A(:, [2 4 5]) = B(:, 1:3)
```
replaces columns 2, 4, and 5 of matrix A with the first three columns of matrix B. Note that the entire altered matrix A is printed and assigned.
```
A =

     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16

>> B = [A*2 A * -1]

B =
     2     4     6     8    -1    -2    -3    -4
    10    12    14    16    -5    -6    -7    -8
    18    20    22    24    -9   -10   -11   -12
    26    28    30    32   -13   -14   -15   -16

>>
>> A(:,[2 4 5]) = B(:, 1:3)

A =

     1     2     3     4     6
     5    10     7    12    14
     9    18    11    20    22
    13    26    15    28    30

>>
```

4.12 Deletion of Rows and Columns

You can delete rows and columns from a matrix using just a pair of square brackets.

```
>> X = M

X =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

Then, to delete the 2nd column of X, use

```
   X(:, 2) = [ ]
```

This changes X to

```
X =

    16     3    13
     5    10     8
     9     6    12
     4    15     1


A(3, :) = []        %delete 3rd row

A(:, 2:4) = []      %deletes 2nd - 4th column

A([1 4], :) = []    % deletes 1st and 4th rows
```

If you delete a single element from a matrix, the result isn't a matrix anymore. So, expressions like

```
   X(1,2) = []
```

result in an error.

```
   >> X(1,2) = []
```

??? Indexed empty matrix assignment is not allowed.

However, using a single subscript deletes a singe element or sequence of elements, and reshapes the remaining elements into a row vector.

```
   X(2:2:10) = []
```

Delete elements starting with the 2nd element; spacing of 2; ending with the 10th element. MAT-LAB is column-major form, i.e., it looks at columns NOT rows.

16 (elem 1)	3 (elem 5)	13 (elem 9)
5 (elem 2)	10 (elem 6)	8 (elem 10)
9 (elem 3)	6 (elem 7)	12 (elem 11)
4 (elem 4)	15 (elem 8)	1 (elem 12)

```
results in
   >> X(2:2:10) = []

   X =

       16      9      3      6     13     12      1
```

4.13 Reversal

```
>> A = [6 9 4 0 3 ; 1 5 7 0 0]

A =

     6     9     4     0     3
     1     5     7     0     0

>> B = A(:,5:-1:1)            % reverses order of columns

B =
     3     0     4     9     6
     0     0     7     5     1

>>
```

4.14 Functions for Handling Arrays

MATLAB has multiple built-in functions for managing and handling arrays.

Function	Description	Example

length(A)	`vector: Returns the number of elements in vector A`	`>> A = [2 4 6];`
		
		`>> length(A)`
		`ans =`
		`       3.00`

size(A)	`row vector [m, n] containing the number of rows:  m and the number of cols: n in matrix A`	`>> A = [2 4 6];`
		`>> size(A)`
		`ans =`
		`     1.00     3.00`
		
		`>> B = [6 -5; -10 0; 3 2];`
		`>> size(B)`
		`ans =`
		`          3.00          2.00`

max(A)
 	
max(B)
 	
[Y,I] = max(C)
	`vector: largest elem in A`
	
	`array: row vector containing max elem of each col`
	
	`returns the indices of the max values in vector`
		`>> A = [2 4 6];`
		`>> max(A)`
		`ans =`
		`          6.00`
		`B =`
		`          6.00         -5.00`
		`        -10.00             0`
		`          3.00          2.00`
		
		`>> B = [6 -5; -10 0; 3 2];`
		`>> max(B)`
		`ans =`
		`          6.00          2.00`

Function	Description	Example

min(A)	`vector: smallest elem in A`	`>> A = [2 4 6];`
		`>> min(A)`
		`ans =`
		`2.00`

min(B)	`array: row vector containing min elem of each col`	`>> B = [6 -5; -10 0; 3 2];`
		`>> min(B)`
		`ans =`
		`-10.00    -5.00`

[Y,I] = min(C)	`returns the indices of the min values in vector`	

mean(A)	`vector: returns average or mean value of elements`	`>> A = [2 4 6];`
		`>> mean(A)`
		`ans =`
		`4`

mean(B)	`array: a row vector containing the mean value of each column`	`>> B = [6 -5; -10 0; 3 2];`
		`>> mean(B)`
		`ans =`
		`-0.3333   -1.0000`

median(A)	`vector: median value of the elements of A`	`>> A = [2 4 6];`
		`>> median(A)`
		`ans =`
		`4`

median(B)	`array: row vector containing median value of each column`	`>> B = [6 -5; -10 0; 3 2];`
		`>> median(B)`
		`ans =`
		`3     0`

std(A)	`vector: returns the standard deviation`
	
	`array: is a row vector containing the standard deviation for each column`
		`>> A = [2 4 6];`
		`>> std(A)`
		`ans =`
		`2`
		
		`>> std(B)`
		`ans =`
		`8.5049    3.6056`

sum(A)	`vector: sums elements`	`>> A = [2 4 6];`
		`>> sum(A)`
		`ans =`
		`12.00`

sum(B)	`array: sums elements of each col of array A; returns a row vector`	`>> B = [6 -5; -10 0; 3 2];`
		`>> sum(B)`
		`ans =`
		`-1.00     -3.00`

Function	Description	Example

sort(A)
 	
sort(B)
	vector: sorts elements in ascending order
	
	array: sorts each column in ascending order
		`>> A = [2 6 4];`
		`>> sort(A)`
		`ans =`
		`     2     4     6`
		
		`>> sort(B)`
		`ans =`
		`   -10    -5`
		`     3     0`
		`     6     2`

det(A)	returns the determinant of a square matrix use COND instead of DET to test for matrix singularity	
dot(A,B)	vector: dot product returns the scalar product of vectors A and B. A & B must be vectors of the same length	
cross(A,B)	vector cross product returns the cross product of the vectors A and B A & B must be 3 element vectors	
inv(A)	the inverse of a square matrix. This is SLOW to use. IT is advised using left division instead.	
cat(dim, A, B)	concatenate arrays A and B along the dimension DIM	cat(2,A,B) same as [A,B] cat(1,A,B) same as [A;B]

Function	Description	Example

find	find indices of nonzero elements
	
	find indices of elements > 3
	
	find nonzero element in an array
		`>> A = [1 0 2 0 0 5];`
		`>> find(A)`
		`ans =`
		`     1     3     6`
		
		`>> I = find(A > 3)`
		`I =`
		`     6`
		
		`B =  1     2     3`
		`     0     4     5`
		`     0     0     6`
		`>> [i j] = find(B)`
		`i =          j =`
		`     1           1`
		`     1           2`
		`     1           3`
		`     2           2`
		`     2           3`
		`     3           3`

end	last index	`X(3:end)`
		`X(1,1:2:end-1)`
		to grow an array
		`X(end+1) = 5`

reshape(A,m,n)	returns the M-by-N matrix whose elements are taken columnwise from X. An error results if X does not have M*N elements.	`>> A = [1 2 3;4 5 6]`
		`>> reshape(A,3,2)`
		`ans =`
		`     1     5`
		`     4     3`
		`     2     6`

diag(V)	When V is a vector, cre-ates a square matrix with the elements of V in the diagonal	`>> V = [1 3 5];`
		`>> diag(V)`
		`ans =`
		`     1     0     0`
		`     0     3     0`
		`     0     0     5`

diag(A)	vector: create matrix with v on the diag
	
	matrix, creates a vector from the diagonal ele-ments of A
		`>> A = [1 2 3;4 5 6;7 8 9];`
		`>> diag(A)`
		`ans =`
		`     1`
		`     5`
		`     9`

Function	Description	Example

triu(A)	`Extract the upper triangular part of a matrix`	`>> A = [1 2 3;4 5 6;7 8 9]`
		`>> U = triu(A)`
		`U =`
		`1 2 3`
		`0 5 6`
		`0 0 9`

triu(A,n)	`Elements on and above the n-th diagonal of X.  n = 0 is the main diagonal. n > 0 above the main diag`
	`n < 0 is below the main diag`
		`>> triu(A,1)`
		`ans =`
		`0 2 3`
		`0 0 6`
		`0 0 0`

tril(A)	`Extract the lower triangular part of a matrix`	`>> A = [1 2 3;4 5 6;7 8 9]`
		
		`> A = tril(A)`
		`A =`
		`1 0 0`
		`4 5 0`
		`7 8 9`

tril(A,n)	`Elements on and below the n-th diagonal of X.  n = 0 is the main diagonal. n > 0 above the main diag`
	`n < 0 is below the main diag`
		`>> tril(A,-1)`
		`ans =`
		`0 0 0`
		`0 0 0`
		`7 0 0`

rot90	`rotates matrix 90 degrees`	
fliplr	`flips from left to right`	
flipud	`flips from up to down`	

Random Numbers		
Function	**Description**	**Example**

`rand`	`uniformly distributed pseudo-random number between 0 and 1.  A sequence of numbers generated is determined by the state of the generator.`	`>> rand`
		`ans =`
		`0.9501`
		`>> rand`
		`ans =`
		`0.2311`

Random Numbers

Function	Description	Example

`rand(1,n)`	generates a row vector of n random numbers	`>> rand(1,5)`
		`ans =`
		`0.6068  0.4860  0.8913`
		`0.7621  0.4565`

`rand(n)`	generates an n by n matrix of random numbers-uniform distribution on interval (0.0, 1.0)	`>> rand(3)`
		`ans =`
		`0.0185    0.6154    0.7382`
		`0.8214    0.7919    0.1763`
		`0.4447    0.9218    0.4057`

`rand(m,n)`	generates an m by n matrix with random numbers on interval (0.0,1.0)	`>> rand(2,3)`
		`ans =`
		`0.9355    0.4103    0.0579`
		`0.9169    0.8936    0.3529`

`randn`
`randn(1,n)`
`randn(n)`
`randn(m,n)`
	normally distributed ran-dom numbers
	
	see above: rand	

`sprand`	sparse uniformly distrib-uted random matrix	

randperm(n)	random permutation of integers from 1 to n	`>> randperm(4)`
		`ans =`
		`2     3     4     1`

permute(A, order)	rearranges the dimensions of A so that they are in the order specified by the vector ORDER	

4.15 Array Arithmetic

The following matrix operators are available in MATLAB:

	MATLAB help precedence		**C++**
Done First ↑	`( )`	parentheses function call	`( )` `function call`
	`^ .^ '`	exponentiation transpose	`NA`
	`~`	not unary plus minus	`!` `unary - +`
	`* .* / ./ \ .\`	multiply, divide, modulus	`* / %`
	`+ -`	binary addition subtraction	`+ -`
	`:`	colon	`NA`
	`< <= ==` `> >= ~=`	relational operators	`< <=` `> >=`
	`see above`	ck for equality not equal	`== !=`
	`&`	element-wise logical AND	`NA`
	`|`	element-wise logical OR	`NA`
	`&& (short-circuit logical AND)`		`and`
	`|| (short-circuit logical OR)`		`or`
↓ Done Last	`=`	assignment	`=`

Note: We strongly recommend that you add parentheses to all expressions using '&' and '|' to avoid any potential problems with the interpretation of your code between different version of MATLAB. Earlier version of MATLAB interpreted these differently than the current version.

These matrix operations apply, of course, to scalars (one-by-one matrices) as well. If the sizes of the matrices are incompatible for the matrix operation, an error message will result, except in the case of scalar-matrix operations (for addition, subtraction, and division as well as for multiplication) in which case each entry of the matrix is operated on by the scalar.

4.15.1 Transpose Operator

The transpose operator, when applied to a vector, switches a row vector to a column vector and a column vector to a row vector. When applied to a matrix, it switches the rows to columns and the columns to rows. The transpose operator is applied to typing a single quote ‘ following the variable to be transposed.

Original	Transpose
`>> A = [1 3 5 7]` `A =` `1 3 5 7`	`>> A'` `ans =` `1` `3` `5` `7`
`A =` `2` `4` `6`	`>> A'` `ans =` `2 4 6`
`>> A = [1 2; 3 4]` `A =` `1 2` `3 4`	`>> A'` `ans =` `1 3` `2 4`
`>> A = [0 1 2;3 0 4;5 6 0]` `A =` `0 1 2` `3 0 4` `5 6 0`	`>> A'` `ans =` `0 3 5` `1 0 6` `2 4 0`

4.15.2 Matrix Addition and Subtraction

If A is a (m-by-n) matrix and B is a (p-by-q) matrix, then the matrix sum C = A + B is defined only when m = p and n = q. The matrix sum is a (m-by-n) matrix C whose elements are

$$c_{ij} = a_{ij} + b_{ij}$$

If $A = \begin{bmatrix} 2 & 1 \\ 4 & 6 \end{bmatrix}$ and $B = \begin{bmatrix} 4 & 2 \\ 0 & 1 \end{bmatrix}$ then C = A + B = $\begin{bmatrix} 2 & 1 \\ 4 & 6 \end{bmatrix} + \begin{bmatrix} 4 & 2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 3 \\ 4 & 7 \end{bmatrix}$

In MATLAB,

```
>> A = [2 1;4 6]
>> B = [4 2;0 1];

>> C = A + B

C =

     6     3
     4     7
```

4.15.3 Matrix subtraction is parallel to matrix addition

```
>> A = [2 1;4 6]
>> B = [4 2;0 1];

>> D = A - B

D =

    -2    -1
     4     5
```

4.15.4 Dot Product

If X = [x1 x2 x3 ... xn]   a row vector containing n elements and

$Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \dots \\ y_n \end{bmatrix}$ is a column vector containing the same number of elements. The dot product (sometimes called scalar product or inner product) is a special case of matrix multiplication and is defined as:

$$X \times Y = \sum_{i=1}^{n} x_i \cdot y_i$$

Example: Dot product of X = [1 2 3 4 5] with Y = X' is

```
X * Y = [1 2 3 4 5] * [1 2 3 4 5]'
      = 1*1 + 2*2 + 3*3 + 4*4 + 5*5
      = 1  + 4 + 9 + 16 + 25
      = 55
```

```
>> X = [1 2 3 4 5];
>> Y = X';

>> X * Y

ans =

    55
```

4.15.5 Matrix Multiplication

Given the two arrays A (m by n) and B (p by q). The matrix product A * B is defined only when the interior matrix dimensions are the same (i.e., n = p). The matrix product C = A * B is a (m by q) matrix whose elements are

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$$

for i = 1, 2, ...m and j = 1,2,...n. Actually, c_{ij} is the dot product of the ith row of A with the jth column of B.

Example:

C = A * B = $A = \begin{bmatrix} 2 & 1 \\ 4 & 6 \end{bmatrix}$ and $B = \begin{bmatrix} 4 & 2 \\ 0 & 1 \end{bmatrix}$ then

$$C = A * B = \begin{bmatrix} 2 \cdot 4 + 1 \cdot 0 & 2 \cdot 2 + 1 \cdot 1 \\ 4 \cdot 4 + 6 \cdot 0 & 4 \cdot 2 + 6 \cdot 1 \end{bmatrix} = \begin{bmatrix} 8 & 5 \\ 16 & 14 \end{bmatrix}$$

```
>> A = [2 1;4 6];
>> B = [4 2;0 1];

>> A * B

ans =

     8     5
    16    14

A * A * A * A is equivalent to A^4

>> A*A*A*A

ans =

         320         384
```

```
        1536        1856

>> A^4

ans =

         320         384
        1536        1856 and the rest of the elements are 0's.  When the iden-
tity matrix multiplies another matrix (or vector), that matrix (or vector) is
unchanged.
A * I = I * A = A
```

4.15.6 Scalar Multiplication of Arrays

Scalar multiplication on matrices is element-by-element. Example:

```
>> 2*A                          % multiply each element of A by 2

ans =

     4     2
     8    12


>> A/3                          %divide each element of A by 3

ans =

    0.6667    0.3333
    1.3333    2.0000


>> 3\A                           %left division of A by 3

ans =

    0.6667    0.3333
    1.3333    2.0000
```

4.15.7 Array Division

The division operation can be explained with the assistance of the identity matrix and the inverse operation.

4.15.8 Identity Matrix:

The identity matrix is a square matrix in which the diagonal elements are 1's and the rest of the elements are 0's. When the identity matrix multiplies another matrix, the matrix remains unchanged.

```
A * I = I * A = A
```

4.15.9 Inverse of a Matrix:

Matrix B is the inverse of matrix A if when two matrices are multiplied the product is the identity matrix. Both matices must be square and the multiplication order can be B*A or A*B

```
B*A = A*B = I
```

The inverse of a matrix A is typically written as A^{-1}. In MATLAB the inverse of a matrix can be obtained either by raising A to the power of -1 or with the **inv(A)** function.

NOTE: finding the inverse of a matrix is very costly (time) in MATLAB. You dont' believe me just try this for something other than a non-trivial array. It is a much better idea to use LEFT DIVISION INSTEAD OF finding the INVERSE.

4.15.10 Array Division:

Left division is used to solve the matrix equation A * X = B. In this equation X and B are column vectors. This equation can be solved by multiplying on the left of both sides by A^{-1}

$$A^{-1} * A * X = A^{-1} * B$$
$$I * X = A^{-1} * B$$
$$X = A^{-1} * B$$

In MATLAB, is written as:

```
X = A \ B
```

NOTE: This is the preferrable means of solving this type of equation. Use left division rather than an inverse.

Right division is used to solve the matrix equation X * C = D. In this equation, X and D are row vectors. This equation is solved by:

$$X * C * C^{-1} = D * C^{-1}$$
$$X * I = D * C^{-1}$$
$$X = D * C^{-1}$$

In MATLAB, $X = D * C^{-1}$ is written as

```
X = D / C
```

4.16 Element-by-Element Operations

There are applications that require operations to be carried out on an element by element basis rather than on an array basis. Addition and subtraction are by definition element-by-element operations. Note element-by-element operations can only be done with arrays of the same size.

Element-by-element operations are entered in MATLAB by typing a period (.) in front of the operator.

Symbol	Element-by-Element
`.*`	multiplication
`.^`	exponentiation
`./`	right division
`.\`	left division

```
>> A = [   2      1;
           4      6];

>> B = [  4      2;
          0      1]


>>
>> A.*B

ans =

     8     2
     0     6


>> A.^B

ans =

    16     1
     1     6

>> A.\B

ans =

    2.0000    2.0000
         0    0.1667

>> A./B
Warning: Divide by zero.
(Type "warning off MATLAB:divideByZero" to suppress this warning.)

ans =

    0.5000    0.5000
       Inf    6.0000
```