

Chapter Two

MATRICES

Def: - A matrix in Matlab is similar to defining a vector commas or spaces are used to separate elements in a row and semi colons are used to separate individual rows.

$$\text{Ex:- } A_1 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
Command Window
>> A1=[1 2 3; 4 5 6; 7 8 9]

A1 =

     1     2     3
     4     5     6
     7     8     9

>> A2=[1:4;-1:2:5]

A2 =

     1     2     3     4
    -1     1     3     5

>> A3=[1 3;-4 7]

A3 =

     1     3
    -4     7
```

Transpose of matrix interchange rows with the corresponding column

```
Command Window
>> A2
A2 =
     1     2     3     4
    -1     1     3     5

>> A2'
ans =
     1    -1
     2     1
     3     3
     4     5
```

To find the dimension of matrix use command **size(A)** where A is matrix

```
Command Window
>> size(A2)
ans =
     2     4

>> size(A2')
ans =
     4     2
```

Special Matrices

`E=[]` % empty 0-by-0 matrix

Command Window

```
>> A=[]
```

```
A =
```

```
 []
```

```
>> I=eye(3)
```

```
I =
```

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

```
>> eye(2,5)
```

```
ans =
```

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

```
>> y=[1 3 -2];
```

```
>> R=diag([y])
```

```
R =
```

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

```

Command Window
>> B=ones(3,2)

B =

     1     1
     1     1
     1     1

>> c=zeros(3,5)

c =

     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0

```

To find determinant of matrix use command **det(A)** where A is matrix

The *determinant* of a square matrix is a number. For a 2×2 matrix, the determinant is given by:

$$D = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$$

To calculate the determinant of a matrix A in MATLAB, simply write **det(A)**. Here is the determinant of a 2×2 matrix:

```

>> A = [1 3; 4 5];
>> det(A)

ans =

    -7

```

```

Command Window
>> A=[1 2 3; 4 5 6; 7 8 9];
>> det(A)

ans =

   -9.5162e-16

```

```
>> trace(A)

ans =

    15

>> sum(diag(A))

ans =

    15
```

```
Command Window
>> A=[1 2 3; 4 5 6; 7 8 9];
>> diag(A)

ans =

     1
     5
     9
```

Diag(A,k)

If $k = 0$ that mean the diagonal of matrix

If $k > 0$ that mean the upper diagonal of matrix

If $k < 0$ that mean the lower diagonal of matrix

```
>> diag(A)
```

```
ans =
```

```
1  
5  
9
```

```
>> diag(A, 0)
```

```
ans =
```

```
1  
5  
9
```

Command Window

```
>> diag(A, 1)
```

```
ans =
```

```
2  
6
```

```
>> diag(A, 2)
```

```
ans =
```

```
3
```

```
>> diag(A, -1)
```

```
ans =
```

```
4  
8
```

```
>> diag(A, -2)
```

```
ans =
```

```
7
```

Example: -

```
>> v=[10 20 30];
>> diag(v)

ans =

    10     0     0
     0    20     0
     0     0    30
```

To find rank use command **rank(A)** where A is matrix

The *rank* of a matrix is a measure of the number of *linearly independent* rows or columns in the matrix. If a vector is linearly independent of a set of other vectors that means it cannot be written as a linear combination of them. Simple example:

```
>> A=[1 2 3; 4 5 6; 7 8 9];
>> rank(A)

ans =

     2
```

To find the value of location of matrix you can do the following

```
Command Window
>> A=[1 2 3; 4 5 6; 7 8 9];
>> A(2,3)

ans =

     6

>> A(3,2)

ans =

     8
```

```
>> A(1,3)=0
```

```
A =
```

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

```
Command Window
```

```
>> A(:,3)
```

```
ans =
```

```
    0
    6
    9
```

```
>> A(2,:)
```

```
ans =
```

```
    4    5    6
```

To find lower and upper triangler of matrix

```
Command Window
```

```
>> A=[1 2 3; 4 5 6; 7 8 9];
```

```
>> tril(A)
```

```
ans =
```

```
    1    0    0
    4    5    0
    7    8    9
```

```
>> tril(A,1)
```

```
ans =
```

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



```
>> tril(A,2)
```

```
ans =
```

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

Command Window

```
>> tril(A,-1)
```

```
ans =
```

```
    0    0    0
    4    0    0
    7    8    0
```

```
>> tril(A,-2)
```

```
ans =
```

```
    0    0    0
    0    0    0
    7    0    0
```

Variable

- 1- they may not start with a numeral
- 2- matlab is case sensitive **A** and **a** are different variables
- 3- we cannot use the words such as **for**, **if**, **switch**, **else**, **if**, ...

Mathematical matlab command

Mathematical Expression	Matlab Command
$\sin x, \cos x$	Sin(x), cos(x)
$\tan^{-1} x$	atan(x)
$\sinh x$	sinh(x)
e^x	exp(x)
Reminder	rem(a,b)
True for real number	isreal(x)
x^b	x^b
\sqrt{x}	sqrt(x) or x^0.5
$ x $	abs(x)
$\ln x, \log x$	log(x), log10(x)
π	pi
$a \pm b$	a±b
ab	a*b
Round toward zero	fix(x)
Round toward $-\infty$	floor(x)
Round toward $+\infty$	ceil(x)
Round to nearest integer	round(x)

Description

`r = rem(a,b)` returns the remainder after division of a by b, where a is the dividend and b is the divisor.

```
>> rem(7,3)           3+3+1 here 1 is reminder
ans =
     1

>> rem(3,7)           0+3 here 3 is reminder
ans =
     3
```

```
Command Window
>> isreal(pi)

ans =

    logical

     1

>> isreal(1+i*8)

ans =

    logical

     0
```

```
Command Window
>> fix(4.5)

ans =

     4

>> fix(4.6)

ans =

     4

>> fix(4.4)

ans =

     4
```

```
>> round(5.4)

ans =

     5

>> round(5.5)

ans =

     6
```

To find maximum or minimum or summation use the following commands

Command Window

```
>> A

A =

     1     2     3
     4     5     6
     7     8     9

>> max(A)

ans =

     7     8     9

>> min(A)

ans =

     1     2     3

>> sum(A)

ans =

    12    15    18
```

```
>> sum(sum(A))  
  
ans =  
  
    45
```

flip

`flip(A)` returns array B the same size as A, but with the order of the elements reversed. If A is a matrix, then `flip(A)` reverses the elements in each column.

```
A =  
  
    1    2    3  
    4    5    6  
    7    8    9  
  
>> flipud(A)  
  
ans =  
  
    7    8    9  
    4    5    6  
    1    2    3  
  
>> fliplr(A)  
  
ans =  
  
    3    2    1  
    6    5    4  
    9    8    7
```

rot90

rot90(A) rotates array A counterclockwise by 90 degrees. For multidimensional arrays, rot90 rotates in the plane formed by the first and second dimensions.

```
>> rot90(A)
```

```
ans =
```

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

```
>> rot90(A,2)
```

```
ans =
```

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

```
>> rot90(A,4)
```

```
ans =
```

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

Sort of matrix

sort(A, direction) sorts the elements of A in ascending order

```
Command Window
>> A=[4 6 2;7 2 9;8 1 6];
>> sort(A, 'descend')

ans =

     8     6     9
     7     2     6
     4     1     2

>> sort(A, 'ascend')

ans =

     4     1     2
```

Command (Find(x))

`find(X)` returns a vector containing the [linear indices](#) of each nonzero element in array X.

```
Command Window
>> v=[1 5 7 0 9];
>> find(v)

ans =

     1     2     3     5

>> A

A =

     4     6     2
     7     2     9
     8     1     6

>> find(A)

ans =

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

```
>> B=[4 6 2;7 2 0;8 0 0];
>> find(B)

ans =

     1
     2
     3
     4
     5
     7
```

Command (all(A))

`all(A)` tests along the first array dimension of `A` whose size does not equal 1, and determines if the elements are all nonzero or logical 1 (true). In practice, `all` is a natural extension of the logical AND operator.

Command Window

```
>> v=[1 5 7 0 9];
>> all(v)

ans =

    logical

     0
```

```
>> v=[0 0 0 0];
>> all(v)

ans =

    logical

     0
```

```
>> v=[1 3 5 7];
>> all(v)

ans =

    logical

     1
```

```
>> B=[4 6 2;7 2 0;8 0 0];
>> all(B)

ans =

    1×3 logical array

     1     0     0
```


Command (any(A))

any(A) tests along the first array dimension of A whose size does not equal 1, and determines if any element is a nonzero number or logical 1 (true). In practice, any is a natural extension of the logical OR operator

```
Command Window
>> B=[4 6 2;7 2 0;8 0 0];
>> any(B)

ans =

    1×3 logical array

     1     1     1

>> v=[0 0 0 0];
>> any(v)

ans =

    logical

     0
```

Command (mean “Average or mean value of array”)

mean(A) returns the mean of the elements of A along the first array dimension whose size does not equal 1.

If A is a vector, then mean(A) returns the mean of the elements.

If A is a matrix, then mean(A) returns a row vector containing the mean of each column.

```
Command Window
>> A
A =
     4     6     2
     7     2     9
     8     1     6

>> mean(A)
ans =
     6.3333     3.0000     5.6667

>> v=[1 3 5 7];
>> mean(v)
ans =
     4
```

Command (factorial)

`factorial(n)` returns the product of all positive integers less than or equal to n , where n is a nonnegative integer value. If n is an array, then f contains the factorial of each value of n . The data type and size of f is the same as that of n .

```
>> factorial(5)

ans =

    120
```

Command (Dot product)

`dot(A,B)` returns the scalar dot product of A and B .

If A and B are vectors, then they must have the same length.

```
Command Window

>> w=[1 2 3 4 5 6];
>> v=[7 8 5 4 3 1];
>> dot(w,v)

ans =

    75
```

Command (Cross product)

`cross(A,B)` returns the cross product of A and B . must be 3 dim mean $A=[a1]$

Create two 3-D vectors.

```
A = [4 -2 1];
B = [1 -1 3];
```

Find the cross product of A and B . The result, C , is a vector that is perpendicular to both A and B .

```
C = cross(A,B)
```

```
C =

    -5    -11    -2
```

Command (primes)

`primes(n)` returns a row vector containing all the prime numbers less than or equal to `n`. The data type of `p` is the same as that of `n`.

```
Command Window
>> primes(30)

ans =

    Columns 1 through 8

     2     3     5     7    11    13    17    19

    Columns 9 through 10

    23    29
```

isprime(X)

`isprime(X)` returns a logical array the same size as `X`. The value at `TF(i)` is true when `X(i)` is a prime number. Otherwise, the value is false

```
>> isprime(30)

ans =

    logical

     0

>> isprime(31)

ans =

    logical

     1
```

Building matrices and extracting part of matrices

Example 1:-

```
>> x=[4;-1];y=[-1,4];
>> X=[x y']
```

X =

```
     4     -1
    -1      4
```

Command Window

```
>> T=[-1 3 4;4 5 6];
>> t=1:3;
>> T1=[T;t]
```

T1 =

```
    -1     3     4
     4     5     6
     1     2     3
```

Command Window

```
>> G=[1 5;4 5;0 2];
>> T2=[T,G']
```

T2 =

```
    -1     3     4     1     4     0
     4     5     6     5     5     2
```

```
>> T2=[T1 G]
```

T2 =

```
    -1     3     4     1     5
     4     5     6     4     5
     1     2     3     0     2
```

```
Command Window
>> T

T =

    -1     3     4
     4     5     6

>> T2

T2 =

    -1     3     4     1     5
     4     5     6     4     5
     1     2     3     0     2

>> G'

ans =

     1     4     0
     5     5     2

>> T3=[T' T2 G]

T3 =

    -1     4    -1     3     4     1     5     1     5
     3     5     4     5     6     4     5     4     5
     4     6     1     2     3     0     2     0     2

|
```

```
Command Window
>> T4=[G',diag([5;6]);ones(3,2),T1]

T4 =

     1     4     0     5     0
     5     5     2     0     6
     1     1    -1     3     4
     1     1     4     5     6
     1     1     1     2     3
```

Example 2: -

```
Command Window
>> A=[ 1 2 3;4 5 6;7 8 9];
>> A(2,3)=15

A =

     1     2     3
     4     5    15
     7     8     9

>> A(2,3)=A(1,3)*A(2,2)

A =

     1     2     3
     4     5    15
     7     8     9
```

Change column 2 by $v=[11 \ 12 \ 13]$

```
>> A(:,2)=[11 12 13]
```

```
A =
```

```
    1    11     3
    4    12    15
    7    13     9
```

```
>> A(3,[1,3])=[18 19]
```

```
A =
```

```
    1    11     3
    4    12    15
   18    13    19
```

Command Window

```
>> A(1:2,[1,3])
```

```
ans =
```

```
    1     3
    4    15
```



```
>> A([1,3],2:3)

ans =

    11     3
    13    19

>> A(2,:)=[]

A =

     1    11     3
    18    13    19

>> A(:,3)=[]

A =

     1    11
    18    13
```

Example: -

```
Command Window

>> A=[ 1 2 3;4 5 6;7 8 9]

A =

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

Swap R1 and R2

```

Command Window
>> A=[ 1 2 3;4 5 6;7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9

>> D=A(1,:);
>> A(1,:)=A(2,:);
>> A(2,:)=D

A =

     4     5     6
     1     2     3
     7     8     9

```

Change C2 with C3

```

>> D=A(:,2);
>> A(:,2)=A(:,3);
>> A(:,3)=D

A =

     4     6     5
     1     3     2
     7     9     8

```

Swap R1 and R3

```

Command Window
>> D=A(1,:);
>> A(1,:)=A(3,:);
>> A(3,:)=D

A =

     5     2     4
     1     3     6
     7     9     5

```

Matrices Arithmetic

```
Command Window
>> A
A =
     5     2     4
     1     3     6
     7     9     5

>> B=[7 12 4;15 19 7;6 4 9]
B =
     7    12     4
    15    19     7
     6     4     9

>> A*B
ans =
    89    114    70
    88     93    79
   214    275   136
```

```
>> A-B
ans =
    -2   -10     0
   -14   -16    -1
     1     5    -4

>> A+B
ans =
    12    14     8
    16    22    13
    13    13    14

>> A/B
ans =
   -0.9456    0.6224    0.3807
    1.3897   -0.8731    0.7281
    0.3202    0.2205    0.2417

>> A.^5
ans =
    3125         32    1024
         1        243   7776
   16807    59049   3125
```

Euclidean Norm

The Euclidean norm (or 2-norm) of a vector v that has N elements is defined by

$$\|v\| = \sqrt{\sum_{k=1}^N |v_k|^2} .$$

\

```
>> v=[ 1 2 3]
```

```
v =
```

```
    1    2    3
```

```
>> norm(v)
```

```
ans =
```

```
    3.7417
```

Magic Function

Matlab has build in function that create magic sequence of almost any size

>> magic(n) → build a square matrix of size $n * n$ and that elements between 0 → n^2

Command Window

```
>> magic(3)
```

```
ans =
```

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

```
>> magic(4)
```

```
ans =
```

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