MATRIX OPERATIONS

Vectors and Matrices

The row vectors:

The row vectors are entities enclosed in pair of square-brakets with numbers seperated either by spaces or by commas. For example, enter two vectors Z and Y as:

```
>> Z = [2,4,6,8]
```

Z =2 4 6 8

```
>> Y = [4 -3 5 -2]
```

Y =4 -3 5 -2

Adding Two Row Matrices:

Z = [2 4 6 8]
Y = [4 -3 5 -2]

>>X=Z+Y

It creates new row matrix

X=
6 1 11 6

Combination of vectors to form another vector:

W =[Z,Y]

W=
2 4 6 8 4 -3 5 -2

The linear combination of Z and Y:

```
>> C=4*Z+6*Y

 C =32    -2    54    20
```

Application of Build in functions and some command to row Matrix:

```
sort(W)

ans = -3    -2     2     4     4     5     6     8
```

```
min(W)

ans =-3
```

```
max(W)

ans =8
```

```
sum(W)

ans =24
```

```
sin(sum(W))

ans =-0.9056
```

```
mean(W)

ans =3
```

```
log(W)

ans =

  Columns 1 through 3

   0.6931         1.3863         1.7918

  Columns 4 through 6

   2.0794         1.3863         1.0986 + 3.1416i

  Columns 7 through 8

   1.6094         0.6931 + 3.1416i
```

Rand: Generates an array with elements randomly chosen from the uniform distribution over the interval [0, 1]

```
rand
ans =
       0.4326
```

Randn: Generates an array with elements randomly chosen from the normal distribution function with zero mean and standard deviation 1.

```
randn
ans =
       0.1253
```

Column Vectors

The column vectors in matlab are formed by using a set of numbers in a pair of square brackets and seperating them with semi-colon. Therefore, one can define two column vectors A and B and add them as below:

```
>> A=[2;3;1],B=[4;6;5], C=A+B

A =
     2
     3
     1
B =
     4
     6
     5
C =
     6
     9
     6
```

Index notation:

```
>> X = [1:9]

X =

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

Or you may not use [ ] notation

```
>> X = 1:9

X =

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

In this example the increment is not given, in this case matlab default value is “1” so matlab understands that the increment is one.

Example :

```
>> X = 0:2:10
X =
    0     2     4     6     8    10
```

In this example, the increment is specified as ‘2’.

Note that in some cases, the upper limit may not be attainable thing.

For example;

```
>> D= 1:0.3:3
D =
   1.0000    1.3000    1.6000    1.9000    2.2000    2.5000    2.8000
```

Example;

```
>> 1:-1:5
ans =
   Empty matrix: 1-by-0
>> 5:-1:1
ans =
    5     4     3     2     1
```

Sections of a Vector:

Let us define a vector using the range notation:

```
>> W=[1:3,7:9]
 W =
     1     2     3     7     8     9
```

Now we would like to extract the middle of two elements of this vector. This can be done with the range notation again. As you can see, the middle two elements are 3:4 range. Therefore, the required part of vector can be obtained as:

```
>> W(3:4)

ans =

     3     7
```

For example; 6:-1:1 is the desending range and when used with part extraction of vector, it gives:

```
>> W(6:-1:1)
ans =
    9     8     7     3     2     1
```

Thus by this way we sorted vector numbers from bigger to smaller values.

```
>>  H=1:2:19
H =
    1     3     5     7     9    11    13    15    17    19
>>  H(3:2:8)
ans =
    5     9    13
```

Example

```
>> M=[3,2,7]; N =[6,8,9]; M+N , M-N , M*N

ans =
     9    10    16

 ans =
    -3    -6    -2

??? Error using ==> mtimes
Inner matrix dimensions must agree
```

As we can see, M+N and M-N are calculated but M*N can not be calculated. Because the summation and substraction can be performed when the marix dimensions are the same. For example, 3x3 matrix can be sum or substract by only 3x3 matrix. But in multiplication, the number of first vector column must be equal to the number of second vector row. For example, (2x3) by (3x1) this results 2x1 vector as shown below.

```
>> M=[3,2,7;2,3,4],  N =[6;8;9],  M*N

M =

     3     2     7
     2     3     4
N =
     6
     8
     9
ans =
    97
    72
```

If you don't want M an N be seen in the window , we can write by using semicolon instead of comma:

```
>>  M=[3,2,7;2,3,4]; N =[6;8;9];M*N

ans =
    97
    72
```

Transpose

We can convert a column vector into a row vector (and vice versa) by a process called transposing denoted by ‘ (single quote).

```
>> c = [ 1; 3; sqrt(5)],c‘

c =
    1.0000
    3.0000
    2.2361

ans =
    1.0000    3.0000    2.2361
```

```
>> w= [3 -5 9], c = [ 1; 3; sqrt(5)],t = w + 2*c'
w =
     3    -5     9
c =
    1.0000
    3.0000
    2.2361
t =
    5.0000    1.0000   13.4721

>> T = 5*w'-2*c
T =
   13.0000
  -31.0000
   40.5279
```

If x is a complex vector, then x' gives the complex conjugate transpose of x:

```
>> x = [1+3i, 2-2i], x’

x =
   1.0000 + 3.0000i   2.0000 - 2.0000i

ans =
   1.0000 - 3.0000i
   2.0000 + 2.0000i
```

Example :

```
>> A=[1,3,6;2,7,8;0,3,9]
```

Then the output appears in the next line as shown below.

```
A =
    1     3     6
    2     7     8
    0     3     9
```

Thus, a matrix is entered row by row, and each row is separated by the semicolon(;). Within each row, elements are separated by a space or a comma(,). Commands and variables used in Matlab are case-sensitive. That is, lower case letters are distinguished from upper case letters. The size of the matrix is checked with

```
>> size(A)

ans =3     3

>> A‘

ans =
    1     2     0
    3     7     3
    6     8     9
```

Arrays and Matrices, Basic Information

Length	Length of vector or largest array dimension
Max	Largest elements in array
Min	Smallest elements in array
Size	Array dimensions
Eye	Identity matrix
Zeros	Create array of all zeros
Cross	Vector cross product
Dot	Vector dot product
Sum	Sum of array elements
Sort	Sort array elements in ascending or descending order
Det	Matrix determinant
Rank	Rank of matrix
Inv	Matrix inverse
Eig	Eigenvalues and eigenvectors
Sqrtm	Matrix square root
Expm	Matrix exponential
Logm	Matrix logarithm

Dot product (dot)

C = dot(A,B)

C = dot(A,B,dim)

C = dot(A,B) returns the scalar product of the vectors A and B. A and B must be vectors of the same length. When A and B are both column vectors, dot(A,B) is the same as A'*B.

For multidimensional arrays A and B, dot returns the scalar product along the first non-singleton dimension of A and B. A and B must have the same size.

C = dot(A,B)

C(:,:,1) =

26 37 174

C(:,:,2) =

91 50 101

C(:,:,3) =

39 75 101

C = dot(A,B,dim) returns the scalar product of A and B in the dimension dim.

Definition: Let a and b be two vectors in Rn, then the dot product of a and b is the scalar a · b given by

a · b = a1b1 + a2b2 + a3b3 + · · · + anbn

C = dot(A,B)

C(:,:,1) =

26 37 174

C(:,:,2) =

91 50 101

C(:,:,3) =

39 75 101

C = dot(A,B,3)

5 1 2-------from A

2 1 5-------from B

\+ _____

10+1+10=21

7 0 1

3 3 2

\+_____

21+0+2=23

8 4 4

6 4 4

\+_____

48+16+16=80

C =

21 23 80

18 66 144

117 73 152

Product of array elements(prod)

D = prod(A)

D = prod(A,dim)

D = prod(A) returns the products along different dimensions of an array.

If A is a vector, prod(A) returns the product of the elements.

If A is a matrix, prod(A) treats the columns of A as vectors, returning a row vector of the products of each column.

If A is a multidimensional array, prod(A) treats the values along the first non-singleton dimension as vectors, returning an array of row vectors.

D = prod(A,dim) takes the products along the dimension of A specified by scalar dim.

$$A = \begin{matrix} 5 & 7 & 8 \\ 0 & 1 & 9 \\ 4 & 3 & 6 \end{matrix}$$

C = prod(A)

5*0*4 7*1*3 8*9*6

C=[0 21 432]

Vector cross product(cross)

C = cross(A,B)

C = cross(A,B,dim)

C = cross(A,B) returns the cross product of the vectors A and B. That is, C = A x B. A and B must be 3-element vectors. If A and B are multidimensional arrays, cross returns the cross product of A and B along the first dimension of length 3.

C = cross(A,B,dim) where A and B are multidimensional arrays, returns the cross product of A and B in dimension dim. A and B must have the same size, and both size(A,dim) and size(B,dim) must be 3.

Example

The cross product of two vectors are calculated as shown:

```
>> a = [1 2 3];

b = [4 5 6];

c = cross(a,b)

c =

    -3     6    -3

c = cross(b,a)
```

Definition: If a = ha1, a2, a3i and b = hb1, b2, b3i, then the cross product of a and b is the vector

a × b = (a2b3 − a3b2, a3b1 − a1b3, a1b2 − a2b1)

NOTE: The cross product is only defined for vectors in R^3.

Matrix determinant(det)

d = det(X)

d = det(X) returns the determinant of the square matrix X. If X contains only integer entries, the result d is also an integer.

The determinant of a matrix **A** is denoted det(**A**), det **A**, or |**A**|.
In the case where the matrix entries are written out in full, the determinant is denoted by surrounding the matrix entries by vertical bars instead of the brackets or parentheses of the matrix. For instance, the determinant of the matrix

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

is written

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

and has the value

$$(aei + bfg + cdh) - (ceg + bdi + afh).$$

Examples

•The statement A = [1 2 3; 4 5 6; 7 8 9]
Produces

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

```
>> A = [1 2 3; 4 5 6; 7 8 9]; det(A)
ans =
     0
```

This happens to be a singular matrix, so d = det(A) produces d = 0. Changing A(3,3) with A(3,3) = 0 turns A into a nonsingular matrix. Now d = det(A) produces d = 27.

•The statement B = [1 7 -5; 4 -3 6; -1 8 9]; det(B)
Produces

```
>> B = [1 7 -5; 4 -3 6; -1 8 9]; det(B)
ans =
  -514
```

Matrix inverse(inv)

Y = inv(X)

Y = inv(X) returns the inverse of the square matrix X. A warning message is printed if X is badly scaled or nearly singular.

In practice, it is seldom necessary to form the explicit inverse of a matrix. A frequent misuse of inv arises when solving the system of linear equations Ax=b.

One way to solve this is with x = inv(A)*b. A better way, from both an execution time and numerical accuracy standpoint, is to use the matrix division operator x = A\b. This produces the solution using Gaussian elimination, without forming the inverse.

Example . Matrix Inverse Operation

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 0 & 1 \\ 1 & -1 & 1 \end{bmatrix}$$

Firstly we create an added unit matrix.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 & 1 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

We add -2 times of first row to second row and -1 times of first row to third row

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & -2 & -1 & -2 & 1 & 0 \\ 0 & -2 & 0 & -1 & 0 & 1 \end{bmatrix}$$

Secos row is divided by -2 and we add the result with times 2 to third row.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1/2 & 1 & -1/2 & 0 \\ 0 & 0 & 1 & 1 & -1 & 1 \end{bmatrix}$$

We add -1/2 times of third row to second row and -1 times of third row to first row

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 1 & -1 \\ 0 & 1 & 0 & 1/2 & 0 & -1/2 \\ 0 & 0 & 1 & 1 & -1 & 1 \end{bmatrix}$$

We add -1 times of second row to first row

$$\begin{bmatrix} 1 & 0 & 0 & -1/2 & 1 & -1/2 \\ 0 & 1 & 0 & 1/2 & 0 & -1/2 \\ 0 & 0 & 1 & 1 & -1 & 1 \end{bmatrix}$$

We obtain inverse matrix of A. Left side is unit matrix. Right side is inverse matrix. ($\mathbf{A}^{-1}$)

$$A^{-1} = \begin{bmatrix} -1/2 & 1 & -1/2 \\ 1/2 & 0 & -1/2 \\ 1 & -1 & 1 \end{bmatrix}$$

```
>> A = [1 2 3; 4 5 6]; B=inv(A)
```

??? Error using ➔ inv

Matrix must be square.

```
>> A = [1 2 3; 4 5 6; 3 4 2]; B=inv(A)
B =
   -1.5556    0.8889   -0.3333
    1.1111   -0.7778    0.6667
    0.1111    0.2222   -0.3333
```

Sum of array elements(sum)

B = sum(A)
B = sum(A,dim)
B = sum(..., ‘double’)
B = sum(..., dim,’double’)
B = sum(..., ‘native’)
B = sum(..., dim,’native’)

B = sum(A) returns sums along different dimensions of an array. If A is a vector, sum(A) returns the sum of the elements.

If A is a matrix, sum(A) treats the columns of A as vectors, returning a row vector of the sums of each column.

If A is a multidimensional array, sum(A) treats the values along the first non-singleton dimension as vectors, returning an array of row vectors.

B = sum(A,dim) sums along the dimension of A specified by scalar dim. The dim input is an integer value from 1 to N, where N is the number of dimensions in A. Set dim to 1 to compute the sum of each column, 2 to sum rows, etc.

B = sum(..., ‘double’) and B = sum(..., dim,’double’) performs additions in double-precision and return an answer of type double, even if A has data type single or an integer data type. This is the default for integer data types.

B = sum(..., ‘native’) and B = sum(..., dim,’native’) performs additions in the native data type of A and return an answer of the same data type. This is the default for single and double.

Remarks:

sum(diag(X)) is the trace of X.

Examples

The magic square of order 3 is

M = magic(3)

M =

8 1 6

3 5 7

4 9 2

This is called a magic square because the sums of the elements in each column are the same.

Sum(M) =

15 15 15

as are the sums of the elements in each row, obtained either by transposing or using thedim argument.

•Transposing

sum(M’) =

15 15 15

•Using the dim argument

sum(M,1)

ans =

15 15 15

Length of vector or largest array dimension(length)

numberOfElements = length(array)

Number Of Elements = length(array) finds the number of elements along the largest dimension of an array. Array is an array of any matlab data type and any valid dimensions. numberOfElements is a whole number of the matlab double class.

For non-empty arrays, number of elements is equivalent to max(size(array)). For empty arrays, number of elements is zero.

Example

Create a 1-by-8 array X and use length to find the number of elements in the second (largest) dimension:

```
X = [5, 3.4, 72, 28/4, 3.61, 17, 94,  89];

length(X)

ans = 8
```

Largest elements in array(max)

C = max(A)
C = max(A,B)
C = max(A,[],dim)
[C,I] = max(...)

C = max(A) returns the largest elements along different dimensions of an array.

If A is a vector, max(A) returns the largest element in A.

If A is a matrix, max(A) treats the columns of A as vectors, returning a row vector containing the maximum element from each column.

If A is a multidimensional array, max(A) treats the values along the first non-singleton dimension as vectors, returning the maximum value of each vector.

C = max(A,B) returns an array the same size as A and B with the largest elements taken from A or B. The dimensions of A and B must match, or they may be scalar.

C = max(A,[],dim) returns the largest elements along the dimension of A specified by scalar dim. For example, max(A,[ ],1) produces the maximum values along the first dimension (the rows) of A.

[C,I] = max(...) finds the indices of the maximum values of A, and returns them in output vector I. If there are several identical maximum values, the index of the first one found is returned.

Smallest elements in array(min)

C = min(A)

C = min(A,B)

C = min(A,[],dim)

[C,I] = min(...)

C = min(A) returns the smallest elements along different dimensions of an array.

If A is a vector, min(A) returns the smallest element in A.

If A is a matrix, min(A) treats the columns of A as vectors, returning a row vector containing the minimum element from each column.

If A is a multidimensional array, min operates along the first nonsingleton dimension.

C = min(A,B) returns an array the same size as A and B with the smallest elements taken from A or B. The dimensions of A and B must match, or they may be scalar.

C = min(A,[],dim) returns the smallest elements along the dimension of A specified by scalar dim. For example, min(A,[],1) produces the minimum values along the first dimension (the rows) of A.

[C,I] = min(...) finds the indices of the minimum values of A, and returns them in output vector I. If there are several identical minimum values, the index of the first one found is returned.

Average or mean value of array (mean)

M = mean(A)
M = mean(A,dim)

M = mean(A) returns the mean values of the elements along different dimensions of an array.

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

If A is a matrix, mean(A) treats the columns of A as vectors, returning a row vector of mean values.

If A is a multidimensional array, mean(A) treats the values along the first non-singleton dimension as vectors, returning an array of mean values.

M = mean(A,dim) returns the mean values for elements along the dimension of A specified by scalar dim. For matrices, mean(A,2) is a column vector containing the mean value of each row.

Examples

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

mean(A)

ans =
    3.0000    4.5000    6.0000

mean(A,2)

ans =
    2.0000
    4.0000
    6.0000
    6.0000
```

Median value of array(median)

M = median(A)

M = median(A,dim)

M = median(A) returns the median values of the elements along different dimensions of an array. A should be of type single or double.

If A is a vector, median(A) returns the median value of A.

If A is a matrix, median(A) treats the columns of A as vectors, returning a row vector of median values.

If A is a multidimensional array, median(A) treats the values along the first nonsingleton dimension as vectors, returning an array of median values.

M = median(A,dim) returns the median values for elements along the dimension of A specified by scalar dim.

Examples

```
A = [1 2 4 4; 3 4 6 6; 5 6 8 8; 5 6 8 8]
A =
    1    2    4    4
    3    4    6    6
    5    6    8    8
    5    6    8    8
median(A)
ans =
    4    5    7    7
median(A,2)
ans =
    3
    5
    7
    7
```

Identity matrix(eye)

Y = eye(n)
Y = eye(m,n)
Y = eye([m n])
Y = eye(size(A))
Y = eye(m, n, classname)
Y = eye(n) returns the n-by-n identity matrix.

Y = eye(m,n) or Y = eye([m n]) returns an m-by-n matrix with 1's on the diagonal and 0's elsewhere. The size inputs m and n should be nonnegative integers. Negative integers are treated as 0.

Y = eye(size(A)) returns an identity matrix the same size as A.

Y = eye(m, n, classname) is an m-by-n matrix with 1's of class classname on the diagonal and zeros of class classname elsewhere. classname is a string specifying the data type of the output. classname can take the following values: 'double', 'single', 'int8', 'uint8', 'int16', 'uint16', 'int32', 'uint32', 'int64', or 'uint64'.

The identity matrix is not defined for higher-dimensional arrays. The assignment y = eye([2,3,4]) results in an error.

Examples

Return a 3-by-5 matrix of class int8:

```
>> x = eye(3,5,'int8')
x =
   1   0   0   0   0
   0   1   0   0   0
   0   0   1   0   0
```

Matrix square root(sqrtm)

X = sqrtm(A)

[X, resnorm] = sqrtm(A)

[X, alpha, condest] = sqrtm(A)

X = sqrtm(A) is the principal square root of the matrix A, i.e. X*X = A.

X is the unique square root for which every eigenvalue has nonnegative real part. If A has any eigenvalues with negative real parts then a complex result is produced. If A is singular then A may not have a square root. A warning is printed if exact singularity is detected.

Example

A matrix representation of the fourth difference operator is

X=[5 -4 1 0 0; -4 6 -4 1 0;1 -4 6 -4 1;0 1 -4 6 -4;0 0 1 -4 5]

X =

5	-4	1	0	0
-4	6	-4	1	0
1	-4	6	-4	1
0	1	-4	6	-4
0	0	1	-4	5

This matrix is symmetric and positive definite. Its unique positive definite square root, Y = sqrtm(X), is a representation of the second difference operator.

Y =

2.0000	-1.0000	0.0000	0.0000	0.0000
-1.0000	2.0000	-1.0000	0.0000	-0.0000
0.0000	-1.0000	2.0000	-1.0000	0.0000
0.0000	0.0000	-1.0000	2.0000	-1.0000
0.0000	-0.0000	0.0000	-1.0000	2.0000

Matrix exponential (expm)

Y = expm(X)

Y = expm(X) computes the matrix exponential of X.

Although it is not computed this way, if X has a full set of eigenvectors V with corresponding eigenvalues D, then

[V,D] = EIG(X) and EXPM(X) = V*diag(exp(diag(D)))/V

Examples

This example computes and compares the matrix exponential of A and the exponential of A.

```
A = [1     1     0
     0     0     2
     0     0    -1 ];

expm(A)
ans =
   2.7183   1.7183    1.0862
   0        1.0000    1.2642
   0         0        0.3679

exp(A)
ans =
   2.7183    2.7183    1.0000
   1.0000    1.0000    7.3891
   1.0000    1.0000    0.3679
```

Notice that the diagonal elements of the two results are equal. This would be true for any triangular matrix. But the off-diagonal elements are different

Matrix logarithm (logm)

L = logm(A)

[L, exitflag] = logm(A)

L = logm(A) is the principal matrix logarithm of A, the inverse of expm(A). L is the unique logarithm for which every eigenvalue has imaginary part lying strictly between $-\pi$ and π. If A is singular or has any eigenvalues on the negative real axis, the principal logarithm is undefined. In this case, logm computes a non-principal logarithm and returns a warning message.

[L, exitflag] = logm(A) returns a scalar exitflag that describes the exit condition of logm:

- If exitflag = 0, the algorithm was successfully completed.