

Vectors and Matrices

1. Input

A **matrix** of size $m \times n$ is a rectangular array which has m rows and n columns. We use the notation $A = (a_{ij})$ for $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$, or in the expanded form:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}.$$

A matrix of size $1 \times n$ with only one row is called a **row vector**; a matrix of size $m \times 1$ is a **column vector**:

$$\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n], \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_m \end{bmatrix}.$$

In Matlab all variables are matrices; even scalars (numbers) are treated as 1×1 matrices. Elements of the matrix may be numbers or valid Matlab expressions. The matrix

$$A = \begin{bmatrix} 2 & -1 & 4.3 \\ 12 & -6.1 & 1.2 \\ 3 & -4.5 & 10 \end{bmatrix}$$

is entered in Matlab by rows:

```
>> clear
```

```
>> A=[2 -1 4.3; 12 -6.1 1.2; 3 -4.5 10]
```

```

A =
    2.0000    -1.0000    4.3000
   12.0000   -6.1000    1.2000
    3.0000   -4.5000   10.0000

```

or

$$B = \begin{bmatrix} e^{-x-2y} \ln x + \operatorname{acos} y & \\ 2-3i & \frac{1-x}{\operatorname{cosh} y} \end{bmatrix}$$

```

>> x=5; y=-.7;
>> B=[exp(-x-2*y) log(x)+acos(y); ...
2-3i (1-x)/cosh(y)]
B =
    0.0273                3.9556
    2.0000 - 3.0000i    -3.1868

```

A matrix (or any long expression) can be entered across multiple lines using three consecutive periods, as shown in the last input.

Vectors are entered the same way as matrices:

```

>> u=[2 0 8]           % u is a row vector
u =
    2    0    8

>> v=[-3;7;6]        % v is a column vector
v =
   -3
    7
    6

```

To create a vector of numbers over a given range, follow the pattern:

vector=InitialValue:Increment:FinalValue,

```
>> u=-8:4:12
```

```
u =
```

```
    -8    -4     0     4     8    12
```

```
>> v=0:pi/4:pi
```

```
v =
```

```
     0    0.7854    1.5708    2.3562    3.1416
```

```
>> w=0:5 % default increment = 1
```

```
w =
```

```
     0     1     2     3     4     5
```

Note that brackets are not required for this operation. Alternatively,

```
>> u1=linspace(-8,12,6)
```

```
u1 =
```

```
    -8    -4     0     4     8    12
```

`linspace(a,b,n)` generates equally spaced vector of length **n** from **a** to **b**.

2. Indexing

Elements of a matrix/vector are accessed/modified by specifying their row and/or column indices. For instance, the element a_{13} of matrix A is:

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

```
ans =
```

```
    4.3000
```

i.e., the element in the 1st row and 3rd column. Matlab also allows to specify and access a range of rows and columns

```

>> clear
>> A=[1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>> B=A(1:3,2:3)    % how to get submatrix
B =
     2     3
     5     6
     8     9

```

or index vectors with numbers representing the desired rows or columns:

```

>> B=A(1:3,[1 3])    % get columns 1 and 3
B =
     1     3
     4     6
     7     9

```

The colon (:) as a row or column index specifies all rows or columns of a matrix:

```

>> B=A(:,2:3)
B =
     2     3
     5     6
     8     9
>> x=A(1,:)    % 1st row of A
x =
     1     2     3

```

```
>> y=A(:,2)      % 2nd column of A
y =
     2
     5
     8
```

The dimensions (number of rows and columns) of matrices in Matlab are determined automatically, it is not necessary to declare dimensions of \mathbf{B} , \mathbf{x} , \mathbf{y} above. However, the dimensions of an existing matrix may be obtained using the command **size**:

```
>> [m,n]=size(B)
m =
     3
n =
     2
```

Function **size** here returns 2 numbers, they should be enclosed by square brackets on the left-hand side of the equal sign.

3. Manipulations with matrix elements.

The transpose operation interchanges rows with columns for matrices with real elements: if $\mathbf{A} = (a_{ij})$, the transposed matrix $\mathbf{B} = \mathbf{A}^T$ has elements $b_{ij} = a_{ji}$. If \mathbf{A} is complex, then $\mathbf{B} = \overline{\mathbf{A}}^T$ is the conjugate transpose, $b_{ij} = \bar{a}_{ji}$. In Matlab,

```
>> A=[3 4;5 6]
A =
     3     4
     5     6
```

```

>> At=A'
At =
     3     5
     4     6
>> C=[1+2i 3+4i;5+6i 7+8i]
C =
 1.0000 + 2.0000i  3.0000 + 4.0000i
 5.0000 + 6.0000i  7.0000 + 8.0000i
>> Ct=C'
Ct =
 1.0000 - 2.0000i  5.0000 - 6.0000i
 3.0000 - 4.0000i  7.0000 - 8.0000i

```

With vectors:

```

>> x=[1 2 3] % row vector
x =
     1     2     3
>> y=x' % column vector
y=
     1
     2
     3

```

Powerful Matlab indexing feature allows other one-line matrix elements manipulations, such as *reshaping*, *appending*, and *deleting* rows and columns. If matrix A is an $m \times n$ matrix, it can be reshaped into a $p \times q$ matrix, as long as $m \times n = p \times q$:

```

>> clear
>> A=[1 2 3 4;5 6 7 8; 9 10 11 12] % 3x4 matrix

```

```

A =
    1     2     3     4
    5     6     7     8
    9    10    11    12
>> B=reshape(A,4,3)           % B is 4x3 matrix
B =
    1     6    11
    5    10     4
    9     3     8
    2     7    12

```

Note, **reshape(A,m,n)** takes elements columnwise from matrix **A**.
 To reshape matrix into a vector:

```

>> clear
>> A=[1 2;3 4]
A =
    1     2
    3     4
>> u=A(:)
u =
    1
    3
    2
    4

```

Clearly, Matlab prefers columns over rows; even with vectors:

```

>> v=[1 2 3 4 5]
    1     2     3     4     5

```

```
>> w=v(:) % the same as w=v'
```

```
w =
```

```
1
2
3
4
5
```

To append row or column to an existing matrix:

```
>> x=[5 6]; A=[A; x] % appends row
```

```
A =
```

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

```
>> y=[7 8 9]'; A=[A y] % appends column
```

```
A =
```

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

Note, the length of the row/column which is appended to the matrix should have the same length as the length of the rows/columns of the existing matrix.

Any row(s) or columns(s) can be easily deleted from a matrix by setting the row or column to a *null* matrix (vector) []:

```
>> A(2,:)=[] % deletes 2nd row
```

```
A =
```

```
1 2 7
5 6 9
```

```
>> A(:,[1 3])=[] % deletes 1st and 3rd columns
```

```
A =
```

```
2
```

```
6
```

```
>> u=1:8 % create vector
```

```
u =
```

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

```
>> u(4:length(u))=[] % deletes 4..8 elements
```

```
u =
```

```
1 2 3
```

```
>> help length % what is length(u)?
```

```
LENGTH Length of vector.
```

```
LENGTH(X) returns the length of vector X. It is equivalent to MAX(SIZE(X)) for non-empty arrays and 0 for empty ones.
```

There are many built-in utility matrices in Matlab, which might be helpful in matrix generation and manipulations. For example,

```
>> zeros(2,3) % 0-matrix
```

```
ans =
```

```
0 0 0
```

```
0 0 0
```

```
>> ones(3,2) % 1-matrix
```

```
ans =
```

```
1 1
```

```
1 1
```

```
1 1
```

```
>> rand(2,4) % random matrix
```

```
ans =
```

```
0.9601 0.7446 0.9334 0.8392
```

```
0.7266 0.2679 0.6833 0.6288
```

These commands with a single argument produce square matrices, e.g,

```
>> rand(3)
ans =
    0.2071    0.3705    0.0439
    0.6072    0.5751    0.0272
    0.6299    0.4514    0.3127
```

A matrix with all elements equal to the same number, say 5, can be created using the function `repmat`:

```
>> repmat(5,2,3)
ans =
     5     5     5
     5     5     5
```

If we put some matrix, e.g., $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ instead of a number:

```
>> repmat([1 2;3 4],2,3)
ans =
     1     2     1     2     1     2
     3     4     3     4     3     4
     1     2     1     2     1     2
     3     4     3     4     3     4
```

EXERCISES

1. Enter the matrix

$$A = \begin{bmatrix} 1 & 4 & 7 & 10 & 13 \\ 2 & 5 & 8 & 11 & 14 \\ 3 & 6 & 9 & 12 & 15 \end{bmatrix}$$

- Compute $A_1 = A^T$.
- Extract the submatrix A_2 of A consisting of columns 1, 3, 5.
- Extract the submatrix of A_1 consisting of the elements in rows 1,2 and 4 and columns 1 and 3.
- Delete rows 3 and 5 of the matrix A^T .
- Interchange rows 1 and 3 of A_2 .
- Explain the results of the following commands:

```
B1=A(3:-1:1, :)
B2=[A [16 17 18]']
A(1:2:12)=[]
```

2. Enter matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \text{ and then obtain the matrix } B = \begin{bmatrix} 10 & 11 & 12 \\ 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix} \text{ from } A.$$

4. Matrix operations and functions.

Matlab allows several high-level operations on matrices of the compatible sizes. The simplest are addition and subtraction:

```
>> clear
>> A=reshape(1:12,3,4)
A =
     1     4     7    10
     2     5     8    11
     3     6     9    12
>> B=reshape(12:-1:1,3,4)
B =
    12     9     6     3
```

```

    11     8     5     2
    10     7     4     1
>> C=A+B
C =
    13    13    13    13
    13    13    13    13
    13    13    13    13
>> D=A-B
D =
   -11    -5     1     7
    -9    -3     3     9
    -7    -1     5    11

```

In the above commands one of the matrices can be replaced with the scalar:

```

>> E=A+100
E =
   101   104   107   110
   102   105   108   111
   103   106   109   112

```

then 100 is added to all elements.

To multiply matrices Matlab uses the multiplication rule “row * column”: if $A = (a_{ij})$ is $m \times n$ matrix and $B = (b_{jk})$ is $n \times l$ matrix, then

their product is $m \times l$ matrix C with elements $c_{ik} = \sum_{j=1}^n a_{ij}b_{jk}$.

```

>> clear
>> A=reshape(1:12,3,4)
A =
     1     4     7    10
     2     5     8    11
     3     6     9    12

```

```
>> B=reshape (8:-1:1,4,2)
```

```
B =
```

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

```
>> C=A*B
```

```
C =
```

```
    128    40
    154    50
    180    60
```

The same rule, applied to vectors is a conventional *scalar* product of two vectors. Since in *Linear Algebra* the “true” vector is a column vector, the scalar product definition of vectors

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ \dots \\ u_n \end{bmatrix} \text{ and } \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix} \text{ is usually written as } \mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T \mathbf{v} = \sum_{i=1}^n u_i v_i.$$

In Matlab,

```
>> u=(1:4)'
```

```
u =
```

```
    1
    2
    3
    4
```

```
>> v=(4:-1:1)'
```

```
v =
```

```
    4
```

```

    3
    2
    1
>> sprod=u'*v
sprod =
    20

```

As a special case, we can multiply matrices and vectors:

```

>> w=A*u   % here A=reshape(1:12,3,4)
w =
    70
    80
    90
>> w'*A
ans =
    500        1220        1940        2660

```

Matlab also provides *element-by-element* operations. If \mathbf{u} and \mathbf{v} are (row) vectors, then

$$\mathbf{u}.*\mathbf{v} \quad \text{produces} \quad [u_1v_1 \ u_2v_2 \ u_3v_3 \ \dots]$$

$$\mathbf{u}./\mathbf{v} \quad [u_1/v_1 \ u_2/v_2 \ u_3/v_3 \ \dots]$$

$$\mathbf{u}.^{\mathbf{v}} \quad [u_1^{v_1} \ u_2^{v_2} \ u_3^{v_3} \ \dots].$$

The same is true for matrices.

```

>> clear; format short
>> u=10*(1:3), v=[5 10 15], u.*v, u./v
u =
    10    20    30

```

```

v =
     5     10     15
ans =
     50    200    450
ans =
     2     2     2
>> 1./v
ans =
     0.2000     0.1000     0.0667
>> w=(1:3)/2, u.^w
w =
     0.5000     1.0000     1.5000
ans =
     3.1623    20.0000   164.3168

```

With matrices:

```

>> clear
>> A=reshape(1:9,3,3), B=10./A, C=A.^B
A =
     1     4     7
     2     5     8
     3     6     9
B =
  10.0000    2.5000    1.4286
   5.0000    2.0000    1.2500
   3.3333    1.6667    1.1111
C =
   1.0000   32.0000   16.1170
  32.0000   25.0000   13.4543
  38.9407   19.8116   11.4887

```

Many built-in math functions can take matrix input and perform element-by-element operations on them.

```
>> sin(A) , sqrt(A) , log(A)
ans =
    0.8415    -0.7568    0.6570
    0.9093    -0.9589    0.9894
    0.1411    -0.2794    0.4121
ans =
    1.0000    2.0000    2.6458
    1.4142    2.2361    2.8284
    1.7321    2.4495    3.0000
ans =
         0    1.3863    1.9459
    0.6931    1.6094    2.0794
    1.0986    1.7918    2.1972
```

The command

```
>> sum(A) % or sum(A,1)
ans =
     6     15     24
```

computes the sum of A elements in columns, i.e., varying the first sub-

script $\sum_i a_{ij}$ for each j -colon. **sum(A,2)** would sum elements in rows.

Similarly, **prod(A)** computes the product of matrix elements.

EXERCISES

1. Generate random square matrices A , B , and C . Check if the associative $(AB)C = A(BC)$ and commutative $AB = BA$ laws of matrix multiplication are satisfied. Show that $(AB)^T = B^T A^T$.

2. Compute ABC , where

$$A = \begin{bmatrix} 991 & 992 & 993 \\ 994 & 995 & 996 \\ 997 & 998 & 999 \\ 1000 & 1001 & 1002 \end{bmatrix}, \quad B = \begin{bmatrix} 12 & -6 & -2 \\ 18 & -9 & -3 \\ 24 & -12 & -4 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 3 & 0 \end{bmatrix}.$$

3. An $n \times n$ matrix A is called convergent if the sequence $\{A^k\} \rightarrow 0$ matrix as $k \rightarrow \infty$.

4. Compute each sum for the values of $n = 50, 500, \dots, 500000$:

a) $1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$;

b) $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots + \frac{(-1)^{n+1}}{n}$,

c) $\frac{1}{2 \cdot 3} - \frac{2}{3 \cdot 4} + \frac{3}{4 \cdot 5} - \frac{4}{5 \cdot 6} + \dots + \frac{(-1)^n n}{(n+1) \cdot (n+2)}$