MATLAB INTRODUCTION

Hyperbolic Trigonometrical Function

```
cosh(0.55)
```

```
ans = 1.1551
```

```
tanh(3)
```

```
ans = 0.9951
```

```
cosh(54)
```

```
ans = 1.4154e+23
```

```
sin(123)
```

```
ans = -0.4599
```

Inverse Hyperbolic

similar to sine cosine

```
asinh(39)
```

```
ans = 4.3569
```

```
atanh(56)
```

```
ans = 0.0179 + 1.5708i
```

Exponential, , and Ln Function

```
%e this will give an error as undefines function
```

e is not recognisable as exponential

```
exp(10)
```

```
ans = 2.2026e+04
```

```
exp(2)
```

```
ans = 7.3891
```

```
2.7183^2
```

```
ans = 7.3892
```

Logarithmic

```
log(3)
```

```
ans = 1.0986
```

this natural log base e, here ln(x)=log(x)

```
log(5)
```

```
ans = 1.6094
```

```
log(10)
```

```
ans = 2.3026
```

for **base 10=log10(x)**

```
log10(1000)
```

```
ans = 3
```

```
log10(10)
```

```
ans = 1
```

```
log10(11.5)
```

```
ans = 1.0607
```

```
log(exp(1))
```

```
ans = 1
```

Introduction to complex numbers

```
1i
```

```
ans = 0.0000 + 1.0000i
```

```
1i
```

```
ans = 0.0000 + 1.0000i
```

i and j are same

```
(2-1i)
```

```
ans = 2.0000 - 1.0000i
```

```
(2-1i)*1i
```

```
ans = 1.0000 + 2.0000i
```

```
(1-4i)-(5-4i)
```

```
ans = -4
```

```
(1-4i)*(1+4i)
```

```
ans = 17
```

```
1i*1i
```

```
ans = -1
```

```
sqrt(-1)
```

```
ans = 0.0000 + 1.0000i
```

```
A=3-1i;
b=7+5i;
A*b
```

```
ans = 26.0000 + 8.0000i
```

```
A=b
```

```
A = 7.0000 + 5.0000i
```

```
A^b
```

```
ans = -1.2891e+05 + 8.9340e+04i
```

abs()

```
C=2;
d=-5;
p=5i;
abs(d)
```

```
ans = 5
```

```
abs(p)
```

```
ans = 5
```

```
abs(b)
```

```
ans = 8.6023
```

```
abs(A)
```

```
ans = 8.6023
```

angle()

angle between complex and real part in radian

```
angle(A)
```

```
ans = 0.6202
```

```
angle(C)
```

```
ans = 0
```

```
angle(d)*180/pi
```

```
ans = 180
```

real()

```
real(A)
```

```
ans = 7
```

imag()

```
imag(A)
```

```
ans = 5
```

```
imag(d)
```

```
ans = 0
```

conj( ) or ( )'

```
conj(A)
```

```
ans = 7.0000 - 5.0000i
```

```
A'
```

```
ans = 7.0000 - 5.0000i
```

complex(x,y)

```
complex(5,6)
```

```
ans = 5.0000 + 6.0000i
```

symbolic toolbox and complex number

```
A
```

```
A = 7.0000 + 5.0000i
```

```
log(A)
```

```
ans = 2.1520 + 0.6202i
```

```
b
```

```
b = 7.0000 + 5.0000i
```

```
exp(A)
```

```
ans = 3.1107e+02 - 1.0516e+03i
```

```
sin(A)
```

```
ans = 48.7549 + 55.9420i
```

```
sym(angle(A))
```

```
ans =
```

$$\frac{5586710707897953}{9007199254740992}$$

```
A=3+3i
```

```
A = 3.0000 + 3.0000i
```

```
sym(angle(A))
```

```
ans =
```

$\frac{\pi}{4}$

```
A=1+1i
```

A = 1.0000 + 1.0000i

```
abs(A)
```

ans = 1.4142

```
sym(abs(A))
```

ans = $\sqrt{2}$

```
(4-5i)*(5+6i)
```

ans = 50.0000 - 1.0000i

```
A=3-4i
```

A = 3.0000 - 4.0000i

```
sym a
```

ans = a

```
A=sym(3-4i)
```

A = $3 - 4\,\mathrm{i}$

```
A*A
```

ans = $-7 - 24\,\mathrm{i}$

```
sqrt(A)
```

ans = $\sqrt{3 - 4\,\mathrm{i}}$

```
A^4-20*A^5+12*A^2
```

ans = $4129 - 62272\,\mathrm{i}$

Vectors in Matlab - Let'slay the foundation

```
A=12
```

```
A = 12
```

```
b=-3
```

```
b = -3
```

```
A=[1 2 3]
```

```
A = 1×3
     1     2     3
```

```
b=[1 4 7]
```

```
b = 1×3
     1     4     7
```

```
C=[3 5 1]
```

```
C = 1×3
     3     5     1
```

```
A=[1 2 3 4 5 6 7]
```

```
A = 1×7
     1     2     3     4     5     6     7
```

```
A=[1,2,3]
```

```
A = 1×3
     1     2     3
```

```
A=[1, 2, 3]
```

```
A = 1×3
     1     2     3
```

Transpose of vector/Matrix

```
A=[1;2;3]
```

```
A = 3×1
     1
     2
     3
```

```
b=[1 2 3;2 3 4;3 2 1]
```

```
b = 3×3
```

```
     1     2     3
     2     3     4
     3     2     1
```

```
b'
```

```
ans = 3×3
     1     2     3
     2     3     2
     3     4     1
```

```
A'
```

```
ans = 1×3
     1     2     3
```

```
C'
```

```
ans = 3×1
     3
     5
     1
```

Extract elemedeents

```
A=[1 2 3 4]
```

```
A = 1×4
     1     2     3     4
```

```
A(2)
```

```
ans = 2
```

```
b=[3 -1 4 5 8 9 7]
```

```
b = 1×7
     3    -1     4     5     8     9     7
```

```
b(2)
```

```
ans = -1
```

```
b(6)
```

```
ans = 9
```

```
b'
```

```
ans = 7×1
     3
    -1
     4
     5
     8
```

```
     9
     7
```

```
A=[2 5 6 8]
```

```
A = 1×4
     2     5     6     8
```

```
b=[3 1 5 9]
```

```
b = 1×4
     3     1     5     9
```

```
A(2)/b(2)
```

```
ans = 5
```

algebra

```
A=[1 2 3 4 5];
b=[5 6 7 8 2];
C=[0 9 6 2 4];
A+b
```

```
ans = 1×5
     6     8    10    12     7
```

```
A-b
```

```
ans = 1×5
    -4    -4    -4    -4     3
```

```
b-A
```

```
ans = 1×5
     4     4     4     4    -3
```

```
A+b+C
```

```
ans = 1×5
     6    17    16    14    11
```

```
%a*c this will give error
```

```
A*C' %this will execute correctly
```

```
ans = 64
```

```
8*A
```

```
ans = 1×5
```

```
     8    16    24    32    40
```

```
-2*A
```

```
ans = 1×5
    -2    -4    -6    -8   -10
```

dot product

```
dot(A,b)
```

```
ans = 80
```

cross product

```
A=[1 2 3];
b=[5 6 7];
cross(A,b)
```

```
ans = 1×3
    -4     8    -4
```

Length of vector

```
A=[1 2 3 5 7]
```

```
A = 1×5
     1     2     3     5     7
```

```
length(A)
```

```
ans = 5
```

```
b=[ 1 2 3 5 6 84 5 6 6]
```

```
b = 1×9
     1     2     3     5     6    84     5     6     6
```

```
length(b)
```

```
ans = 9
```

```
length(A)/length(b)
```

```
ans = 0.5556
```

sum,length,min,max of elements

```
sum(A)
```

```
ans = 18
```

```
b
```

```
b = 1×9
     1     2     3     5     6    84     5     6     6
```

```
sum(b)
```

```
ans = 118
```

```
length(b)
```

```
ans = 9
```

```
sum(b)/length(b)
```

```
ans = 13.1111
```

```
mean(b)
```

```
ans = 13.1111
```

```
min(b)
```

```
ans = 1
```

```
max(b)
```

```
ans = 84
```

Creating vectors

a:b vector a

```
A=2:6
```

```
A = 1×5
     2     3     4     5     6
```

```
C=1:2:8
```

```
C = 1×4
     1     3     5     7
```

```
A=1:8
```

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

```
A(4)
```

```
ans = 4
```

```
A(2:5)
```

```
ans = 1×4
     2     3     4     5
```

linspace

```
help linspace
```

```
 linspace Linearly spaced vector.
    linspace(X1, X2) generates a row vector of 100 linearly
    equally spaced points between X1 and X2.

    linspace(X1, X2, N) generates N points between X1 and X2.
    For N = 1, linspace returns X2.

    Class support for inputs X1,X2:
       float: double, single

    See also logspace, colon.

    Reference page for linspace
    Other functions named linspace
```

```
linspace(1,10)
```

```
ans = 1×100
    1.0000    1.0909    1.1818    1.2727    1.3636    1.4545    1.5455    1.6364 ···
```

```
linspace(1,10,5)
```

```
ans = 1×5
    1.0000    3.2500    5.5000    7.7500   10.0000
```

```
mean(ans)
```

```
ans = 5.5000
```

```
length(ans)
```

```
ans = 1
```

```
A=[1 2 3 6 9 5 6]
```

```
A = 1×7
     1     2     3     6     9     5     6
```

```
b=[3 6 -1 7 1 8 6 7 8]
```

```
b = 1×9
     3     6    -1     7     1     8     6     7     8
```

```
vector1=A(3:5)
```

```
vector1 = 1×3
     3     6     9
```

```
vector2=b(2:7)
```

```
vector2 = 1×6
     6    -1     7     1     8     6
```

```
vector3=[vector1,vector2]
```

```
vector3 = 1×9
     3     6     9     6    -1     7     1     8     6
```

```
vector3=[vector1,vector2]'
```

```
vector3 = 9×1
     3
     6
     9
     6
    -1
     7
     1
     8
     6
```

```
vector4=linspace(1,9,5)
```

```
vector4 = 1×5
     1     3     5     7     9
```

```
vector3=[vector1,vector2,vector4]'
```

```
vector3 = 14×1
     3
     6
     9
     6
    -1
     7
     1
     8
     6
     1
     ⋮
```

```
sum(vector4)
```

```
ans = 25
```

```
sum(vector3)
```

```
ans = 70
```

element wise multiplication(.*) and division(./)

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

```
A = 1×11
     1     2     3     4     5     6     8     9     5     2     7
```

```
b=[7 8 9 5 4 2 3 6 5 2 8]
```

```
b = 1×11
     7     8     9     5     4     2     3     6     5     2     8
```

```
dot(A,b)
```

```
ans = 265
```

```
%a*b it will show error as dimension error
A.*b   %element wise multiplication
```

```
ans = 1×11
     7    16    27    20    20    12    24    54    25     4    56
```

```
A./b    %elementwise division
```

```
ans = 1×11
    0.1429    0.2500    0.3333    0.8000    1.2500    3.0000    2.6667    1.5000 ⋯
```

```
A=[1 2 4 -7 -2 2-9];
b=[1 4 5 6 5 6 8];
abs(A)
```

```
ans = 1×6
     1     2     4     7     2     7
```

```
sin(A)
```

```
ans = 1×6
    0.8415    0.9093   -0.7568   -0.6570   -0.9093   -0.6570
```

```
log(A)
```

```
ans = 1×6 complex
   0.0000 + 0.0000i   0.6931 + 0.0000i   1.3863 + 0.0000i   1.9459 + 3.1416i ⋯
```

```
sqrt(b)
```

```
ans = 1×7
```

```
    1.0000    2.0000    2.2361    2.4495    2.2361    2.4495    2.8284
```

```
tan(A)
```

```
ans = 1×6
    1.5574   -2.1850    1.1578   -0.8714    2.1850   -0.8714
```

```
sinh(b)
```

```
ans = 1×7
10^3 ×
    0.0012    0.0273    0.0742    0.2017    0.0742    0.2017    1.4905
```

```
acos(b)
```

```
ans = 1×7 complex
   0.0000 + 0.0000i   0.0000 + 2.0634i   0.0000 + 2.2924i   0.0000 + 2.4779i ⋯
```

```
%a^2 error
A.^2
```

```
ans =
     1     4    16    49     4    49
```

```
%a^b error
%a.^b
```

Creating vector by Random function

```
rand
```

```
ans = 0.8147
```

```
rand
```

```
ans = 0.9058
```

```
help rand
```

```
rand Uniformly distributed pseudorandom numbers.
   R = rand(N) returns an N-by-N matrix containing pseudorandom values drawn
   from the standard uniform distribution on the open interval(0,1).  rand(M,N)
   or rand([M,N]) returns an M-by-N matrix.  rand(M,N,P,...) or
   rand([M,N,P,...]) returns an M-by-N-by-P-by-... array.  rand returns a
   scalar.  rand(SIZE(A)) returns an array the same size as A.

   Note: The size inputs M, N, P, ... should be nonnegative integers.
   Negative integers are treated as 0.

   R = rand(..., CLASSNAME) returns an array of uniform values of the
   specified class. CLASSNAME can be 'double' or 'single'.

   R = rand(..., 'like', Y) returns an array of uniform values of the
   same class as Y.
```

```
The sequence of numbers produced by rand is determined by the settings of
the uniform random number generator that underlies rand, RANDI, and RANDN.
Control that shared random number generator using RNG.

Examples:

   Example 1: Generate values from the uniform distribution on the
   interval (a, b).
      r = a + (b-a).*rand(100,1);

   Example 2: Use the RANDI function, instead of rand, to generate
   integer values from the uniform distribution on the set 1:100.
      r = randi(100,1,5);

   Example 3: Reset the random number generator used by rand, RANDI, and
   RANDN to its default startup settings, so that rand produces the same
   random numbers as if you restarted MATLAB.
      rng('default')
      rand(1,5)

   Example 4: Save the settings for the random number generator used by
   rand, RANDI, and RANDN, generate 5 values from rand, restore the
   settings, and repeat those values.
      s = rng
      u1 = rand(1,5)
      rng(s);
      u2 = rand(1,5) % contains exactly the same values as u1

   Example 5: Reinitialize the random number generator used by rand,
   RANDI, and RANDN with a seed based on the current time.  rand will
   return different values each time you do this.  NOTE: It is usually
   not necessary to do this more than once per MATLAB session.
      rng('shuffle');
      rand(1,5)

See Replace Discouraged Syntaxes of rand and randn to use RNG to replace
rand with the 'seed', 'state', or 'twister' inputs.

See also randi, randn, rng, RandStream, RandStream/rand,
         sprand, sprandn, randperm.

Reference page for rand
Other functions named rand
```

```
rand(1,4)
```

```
ans =
    0.1270    0.9134    0.6324    0.0975
```

```
rand(1,4)
```

```
ans =
    0.2785    0.5469    0.9575    0.9649
```

```
rand(2,4)
```

```
ans =
    0.1576    0.9572    0.8003    0.4218
    0.9706    0.4854    0.1419    0.9157
```

```
rand(2,4)
```

```
ans =
    0.7922    0.6557    0.8491    0.6787
    0.9595    0.0357    0.9340    0.7577
```

```
sin(rand(1,4))
```

```
ans =
    0.6766    0.3822    0.6095    0.1704
```

```
randperm(5)
```

```
ans =
     2     4     5     3     1
```

```
randperm(100,5)
```

```
ans =
    83    69    32    93     4
```

```
help randperm
```

```
 randperm Random permutation.
    P = randperm(N) returns a vector containing a random permutation of the
    integers 1:N.  For example, randperm(6) might be [2 4 5 6 1 3].

    P = randperm(N,K) returns a row vector containing K unique integers
    selected randomly from 1:N.  For example, randperm(6,3) might be [4 2 5].

    randperm(N,K) returns a vector of K unique values.  This is sometimes
    referred to as a K-permutation of 1:N or as sampling without replacement.
    To allow repeated values in the selection, sometimes referred to as
    sampling with replacement, use RANDI(N,1,K).

    randperm calls RAND and therefore changes the state of the random number
    generator that underlies RAND, RANDI, and RANDN.  Control that shared
    generator using RNG.

    See also nchoosek, perms, rand, randi, rng.

    Reference page for randperm
    Other functions named randperm
```

```
help ones
```

```
 ones   Ones array.
    ones(N) is an N-by-N matrix of ones.

    ones(M,N) or ones([M,N]) is an M-by-N matrix of ones.

    ones(M,N,P,...) or ones([M N P ...]) is an M-by-N-by-P-by-... array of
    ones.

    ones(SIZE(A)) is the same size as A and all ones.

    ones with no arguments is the scalar 1.

    ones(..., CLASSNAME) is an array of ones of class specified by the
    string CLASSNAME.
```

```
    ones(..., 'like', Y) is an array of ones with the same data type, sparsity,
    and complexity (real or complex) as the numeric variable Y.

    Note: The size inputs M, N, and P... should be nonnegative integers.
    Negative integers are treated as 0.

    Example:
       x = ones(2,3,'int8');

    See also eye, zeros.

    Reference page for ones
    Other functions named ones
```

```
ones(3,4)
```

```
ans =
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

```
ones(1,4)
```

```
ans =
     1     1     1     1
```

```
help zeros
```

```
 zeros  Zeros array.
    zeros(N) is an N-by-N matrix of zeros.

    zeros(M,N) or zeros([M,N]) is an M-by-N matrix of zeros.

    zeros(M,N,P,...) or zeros([M N P ...]) is an M-by-N-by-P-by-... array of
    zeros.

    zeros(SIZE(A)) is the same size as A and all zeros.

    zeros with no arguments is the scalar 0.

    zeros(..., CLASSNAME) is an array of zeros of class specified by the
    string CLASSNAME.

    zeros(..., 'like', Y) is an array of zeros with the same data type, sparsity,
    and complexity (real or complex) as the numeric variable Y.

    Note: The size inputs M, N, and P... should be nonnegative integers.
    Negative integers are treated as 0.

    Example:
       x = zeros(2,3,'int8');

    See also eye, ones.

    Reference page for zeros
    Other functions named zeros
```

```
zeros(2,4)
```

```
ans =
     0     0     0     0
```

```
     0     0     0     0
```

```
zeros(1,4)
```

```
ans =
     0     0     0     0
```

```
A=randperm(5)
```

```
A =
     5     2     1     3     4
```

```
help sort
```

```
 sort   Sort in ascending or descending order.
    B = sort(A) sorts in ascending order.
    The sorted output B has the same type and size as A:
    - For vectors, sort(A) sorts the elements of A in ascending order.
    - For matrices, sort(A) sorts each column of A in ascending order.
    - For N-D arrays, sort(A) sorts along the first non-singleton dimension.

    B = sort(A,DIM) also specifies a dimension DIM to sort along.

    B = sort(A,DIRECTION) and B = sort(A,DIM,DIRECTION) also specify the
    sort direction. DIRECTION must be:
        'ascend'  - (default) Sorts in ascending order.
        'descend' - Sorts in descending order.

    B = sort(A,...,'MissingPlacement',M) also specifies where to place the
    missing elements (NaN/NaT/<undefined>/<missing>) of A. M must be:
        'auto'  - (default) Places missing elements last for ascending sort
                  and first for descending sort.
        'first' - Places missing elements first.
        'last'  - Places missing elements last.

    B = sort(A,...,'ComparisonMethod',C) specifies how to sort complex
    numbers. The comparison method C must be:
        'auto' - (default) Sorts real numbers according to 'real', and
                 complex numbers according to 'abs'.
        'real' - Sorts according to REAL(A). Elements with equal real parts
                 are then sorted by IMAG(A).
        'abs'  - Sorts according to ABS(A). Elements with equal magnitudes
                 are then sorted by ANGLE(A).

    [B,I] = sort(A,...) also returns a sort index I which specifies how the
    elements of A were rearranged to obtain the sorted output B:
    - If A is a vector, then B = A(I).
    - If A is an m-by-n matrix and DIM = 1, then
        for j = 1:n, B(:,j) = A(I(:,j),j); end

    The sort ordering is stable. Namely, when more than one element has the
    same value, the order of the equal elements is preserved in the sorted
    output B and the indices I relating to equal elements are ascending.

    Examples:
      % Sort a vector in ascending order
        sort([0 3 1 0 2 0 1 6])
      % Sort each column or row of a matrix
        A = [3 7 5; 0 4 2]
        B1 = sort(A,1)  % sort each column
        B2 = sort(A,2)  % sort each row
      % Sort complex numbers according to their real part
```

```
        A = [1+1j ;  1-1j ;  -2-2j ;  0 ;  -2+2j]
        B = sort(A,'ComparisonMethod','real')

    See also issorted, sortrows, min, max, mink, maxk.

    Reference page for sort
    Other functions named sort
```

```
A=randperm(8)
```

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

```
b=sort(A)
```

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

Statistical Analysis on Vectors

```
data=[10 25 25 58 56 89 42 42 42 5 5 4 ]
```

```
data =
    10    25    25    58    56    89    42    42    42     5     5     4
```

```
mean(data)
```

```
ans = 33.5833
```

```
median(data)
```

```
ans = 33.5000
```

```
std(data)
```

```
ans = 26.2764
```

```
var(data)
```

```
ans = 690.4470
```

```
sqrt(var(data))
```

```
ans = 26.2764
```

Matrices

```
vector1=[2 6 8 0 1]
```

```
vector1 =
     2     6     8     0     1
```

```
vector1(4)
```

```
ans = 0
```

```
matrix1=[1 2;2 3]
```

```
matrix1 =
     1     2
     2     3
```

```
matrix2=[1 3 7;4 8 1;9 5 -2]
```

```
matrix2 =
     1     3     7
     4     8     1
     9     5    -2
```

```
matrix3=[2 5 9;-2 -1 0]
```

```
matrix3 =
     2     5     9
    -2    -1     0
```

```
matrix4=[1 2 7 3;-8 9 5 2;9 6 3 2;7 5 3 4]
```

```
matrix4 =
     1     2     7     3
    -8     9     5     2
     9     6     3     2
     7     5     3     4
```

```
matrix4(3,4)
```

```
ans = 2
```

```
matrix4(6)
```

```
ans = 9
```

```
matrix4(2)
```

```
ans = -8
```

```
matrix4(1:9)
```

```
ans =
     1    -8     9     7     2     9     6     5     7
```

```
matrix4(2:4,1:3)
```

```
ans =
    -8     9     5
     9     6     3
     7     5     3
```

```
b=matrix4(1:2,3:4)
```

```
b =
     7     3
     5     2
```

```
b=matrix4(4,2:4)
```

```
b =
     5     3     4
```

```
matrix4(2:end,1:3)
```

```
ans =
    -8     9     5
     9     6     3
     7     5     3
```

Multiply Matrices and Elementwise Multiplication

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

```
A =
     1     2
     3     4
```

```
b=[5 8;6 1]
```

```
b =
     5     8
     6     1
```

```
A+b
```

```
ans =
     6    10
     9     5
```

```
A-b
```

```
ans =
    -4    -6
    -3     3
```

```
A*b   %matrix multiplication
```

```
ans =
    17    10
    39    28
```

```
A.*b  %element wise multiplication
```

```
ans =
     5    16
```

```
    18     4
```

```
%a+matrix4 (error Mtrix dimension must agree)
pi*A
```

```
ans =
    3.1416    6.2832
    9.4248   12.5664
```

```
3/8*8*sqrt(5)*sym(-5/8*A)
```

ans =

$$\begin{pmatrix} -\frac{15\sqrt{5}}{8} & -\frac{15\sqrt{5}}{4} \\ -\frac{45\sqrt{5}}{8} & -\frac{15\sqrt{5}}{2} \end{pmatrix}$$

```
double ans
```

```
ans =
    97   110   115
```

```
help double
```

```
 double Convert to double precision.
    double(X) returns the double precision value for X.
    If X is already a double precision array, double has
    no effect.

    double is called for the expressions in FOR, IF, and WHILE loops
    if the expression isn't already double precision.  double should
    be overloaded for all objects where it makes sense to convert it
    into a double precision value.

    See also single, datatypes, isfloat, isnumeric.

    Reference page for double
    Other functions named double
```

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

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

```
i*C
```

```
ans =
   0.0000 + 1.0000i   0.000
   0.0000 + 4.0000i   0.000
   0.0000 + 7.0000i   0.000
```

```
(3-8i)*C
```

```
ans =
   3.0000 - 8.0000i   6.000
  12.0000 -32.0000i  15.000
  21.0000 -56.0000i  24.000
```

```
vector1=[1 2 3 5 4 86 6]
```

```
vector1 =
     1     2     3     5     4    86     6
```

```
sum(vector1)
```

```
ans = 107
```

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

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

```
sum(matrix) %return the sum of the column
```

```
ans =
    12    15    18
```

```
sum(ans)
```

```
ans = 45
```

```
sum(sum(matrix))
```

```
ans = 45
```

Min Max element in matrix

```
vector1=[1 2 3 5 4 86 6]
```

```
vector1 =
     1     2     3     5     4    86     6
```

```
min(vector1)
```

```
ans = 1
```

```
max(vector1)
```

```
ans = 86
```

```


A=[1 2 -4 6 8;-9 0 12 -85 9;102 6 9 -23 76]
```

```
A =
     1     2    -4     6     8
    -9     0    12   -85     9
   102     6     9   -23    76
```

```
min(A) %return min of each column
```

```
ans =
    -9     0    -4   -85     8
```

```
min(ans)
```

```
ans = -85
```

```
max(A) %return max of each column
```

```
ans =
   102     6    12     6    76
```

```
max(ans)
```

```
ans = 102
```

```
min(min(A))
```

```
ans = -85
```

```
max(max(A))
```

```
ans = 102
```

size and numel function

```
size(A)
```

```
ans =
     3     5
```

```
numel(A)
```

```
ans = 15
```

augment a matrix

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

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

```
B=[5 6;4 3]
```

```
B =
     5     6
     4     3
```

```
%c=[A;B] ERROR using vertcat.
% dimensions of arrays being
% concatenated are not consistent
```

```
B=[5 6 5;4 3 7]
```

```
B =
     5     6     5
     4     3     7
```

```
C=[A;B]
```

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

important functions for working with Matrices

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

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

```
sin(A)
```

```
ans =
    0.8415    0.9093    0.1411
   -0.7568   -0.9589   -0.2794
   -0.2794    0.6570    0.9894
```

```
sinh(A)
```

```
ans =
    0.0012    0.0036    0.0100
    0.0273    0.0742    0.2017
    0.2017    0.5483    1.4905
```

```
sqrt(A)
```

```
ans =
    1.0000    1.4142    1.7321
    2.0000    2.2361    2.4495
```

```
    2.4495    2.6458    2.8284
```

```
%doc sqrtm
help sqrtm
```

```
 sqrtm     Matrix square root.
    X = sqrtm(A) is the principal square root of the square matrix A.
    That is, X*X = A.

    X is the unique square root for which every eigenvalue has nonnegative
    real part.  If A has any real, negative eigenvalues 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.

    [X, RESNORM] = sqrtm(A) does not print any warning, and returns the
    residual, norm(A-X^2,1)/norm(A,1).

    [X, ALPHA, CONDX] = sqrtm(A) returns a stability factor ALPHA and an
    estimate CONDX of the matrix square root condition number of X, in
    the 1-norm. The residual norm(A-X^2,1)/norm(A,1) is bounded
    approximately by N*ALPHA*EPS and the 1-norm relative error in X is
    bounded approximately by N*ALPHA*CONDX*EPS, where N is the size of
    the matrix.

    See also expm, logm, funm.

    Reference page for sqrtm
    Other functions named sqrtm
```

```
X=sqrtm(A)
```

```
X =
   0.4518 + 0.7500i   0.564
   1.0328 - 0.0000i   1.291
   1.4201 - 0.5000i   1.775
```

```
X*X
```

```
ans =
   1.0000 - 0.0000i   2.000
   4.0000 - 0.0000i   5.000
   6.0000 - 0.0000i   7.000
```

```
exp(A)
```

```
ans =
    0.0027    0.0074    0.0201
    0.0546    0.1484    0.4034
    0.4034    1.0966    2.9810
```

```
expm(A)
```

```
ans =
    0.3814    0.4767    0.5721
    0.8717    1.0897    1.3076
    1.1986    1.4983    1.7980
```

```
log(A)
```

```
ans =
         0    0.6931    1.0986
    1.3863    1.6094    1.7918
    1.7918    1.9459    2.0794
```

```
logm(A)
```

```
Warning: Principal matrix logarithm is not defined for A with nonpositive real eigenvalues. A non-principal
matrix logarithm is returned.
ans =
  -4.6228 + 2.3562i   12.741
  10.5997 - 0.0000i  -23.791
  -3.9458 - 1.5708i   13.588
```

```
logm(expm(A))
```

```
ans =
    1.0000    2.0000    3.0000
    4.0000    5.0000    6.0000
    6.0000    7.0000    8.0000
```

```
log(exp(A))
```

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

```
A^2
```

```
ans =
    27    33    39
    60    75    90
    82   103   124
```

```
A.^2
```

```
ans =
     1     4     9
    16    25    36
    36    49    64
```

Special matrices

```
A=zeros(5)
```

```
A =
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

```
     0     0     0     0     0
     0     0     0     0     0
```

```
ones(5)
```

```
ans =
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

```
zeros(3,4)
```

```
ans =
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

```
ones(3,5)
```

```
ans =
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

```
eye(4)
```

```
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

```
eye(3,5)
```

```
ans =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
```

```
eye(4,3)
```

```
ans =
     1     0     0
     0     1     0
     0     0     1
     0     0     0
```

```
help magic
```

```
 magic  Magic square.
    magic(N) is an N-by-N matrix constructed from the integers
    1 through N^2 with equal row, column, and diagonal sums.
    Produces valid magic squares for all N > 0 except N = 2.
```

```
Reference page for magic
```

```
magic(3)
```

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

```
magic(5)
```

```
ans =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9
```

Transpose of the vector

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

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

```
A'
```

```
ans =
     1     5
     2     6
     3     8
     4     9
```

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

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

```
D=diag(A)
```

```
D =
     1
     5
     8
```

```
D'
```

```
ans =
     1     5     8
```

```
triu(A)
```

```
ans =
     1     2     3
     0     5     6
     0     0     8
```

```
tril(A)
```

```
ans =
     1     0     0
     4     5     0
     6     7     8
```

```
triu(A)+tril(A)
```

```
ans =
     2     2     3
     4    10     6
     6     7    16
```

solve equations using Matrices-

rref(reduced row echelon form) function

```
help rref
```

```
 rref   Reduced row echelon form.
    R = rref(A) produces the reduced row echelon form of A.

    [R,jb] = rref(A) also returns a vector, jb, so that:
        r = length(jb) is this algorithm's idea of the rank of A,
        x(jb) are the bound variables in a linear system, Ax = b,
        A(:,jb) is a basis for the range of A,
        R(1:r,jb) is the r-by-r identity matrix.

    [R,jb] = rref(A,TOL) uses the given tolerance in the rank tests.

    Roundoff errors may cause this algorithm to compute a different
    value for the rank than RANK, ORTH and NULL.

    Class support for input A:
       float: double, single

    See also rank, orth, null, qr, svd.

    Reference page for rref
    Other functions named rref
```

4x+3y=5

5x+8y=3

```
A=[4 3 5;5 8 3]
```

```
A =
```

```
     4     3     5
     5     8     3
```

```
rref(A)
```

```
ans =
    1.0000         0    1.8235
         0    1.0000   -0.7647
```

```
B=[1 3 -2 6;5 6 4 7;1 -23 8 2]
```

```
B =
     1     3    -2     6
     5     6     4     7
     1   -23     8     2
```

```
rref(B)
```

```
ans =
    1.0000         0         0    3.8029
         0    1.0000         0   -0.6350
         0         0    1.0000   -2.0511
```

trace,inverse...

```
A=[10 2 3;7 5 6;6 7 8]
```

```
A =
    10     2     3
     7     5     6
     6     7     8
```

```
trace(A)
```

```
ans = 23
```

```
rank(A)
```

```
ans = 3
```

```
det(A)
```

```
ans = -3.0000
```

```
inv(A)
```

```
ans =
    0.6667   -1.6667    1.0000
    6.6667  -20.6667   13.0000
   -6.3333   19.3333  -12.0000
```

symbolic calculation in matrix

```
syms x A B
A=[2-x 3*x^3;4*x^2-3*x, 9*x^3+2]
```

A =

$$\begin{pmatrix} 2-x & 3\,x^3 \\ 4\,x^2-3\,x & 9\,x^3+2 \end{pmatrix}$$

```
B=[2*x-x^3 8*x+1;x-x^3 5*x+8*x^3]
```

B =

$$\begin{pmatrix} 2\,x-x^3 & 8\,x+1 \\ x-x^3 & 8\,x^3+5\,x \end{pmatrix}$$

```
inv(A)
```

ans =

$$\begin{pmatrix} -\frac{9\,x^3+2}{\sigma_1} & \frac{3\,x^3}{\sigma_1} \\ -\frac{3\,x-4\,x^2}{\sigma_1} & \frac{x-2}{\sigma_1} \end{pmatrix}$$

where

$$\sigma_1 = 2\,\left(6\,x^5-9\,x^3+x-2\right)$$

```
A*B
```

ans =

$$\begin{pmatrix} 3\,x^3\,\left(x-x^3\right)-\sigma_4\,\left(x-2\right) & 3\,x^3\,\sigma_1-\left(8\,x+1\right)\,\left(x-2\right) \\ \left(x-x^3\right)\,\sigma_3-\sigma_4\,\sigma_2 & \sigma_1\,\sigma_3-\left(8\,x+1\right)\,\sigma_2 \end{pmatrix}$$

where

$$\sigma_1 = 8\,x^3+5\,x$$

$$\sigma_2 = 3\,x-4\,x^2$$

$$\sigma_3 = 9\,x^3+2$$

$$\sigma_4 = 2\,x-x^3$$

```
A.*B
```

```
ans =
```

$$\begin{pmatrix} -(2\,x-x^3)\;(x-2) & 3\,x^3\;(8\,x+1) \\ -(x-x^3)\;(3\,x-4\,x^2) & (8\,x^3+5\,x)\;(9\,x^3+2) \end{pmatrix}$$

```
trace(B)
```

ans = $7\,x^3+7\,x$

```
det(A)
```

ans = $-12\,x^5+18\,x^3-2\,x+4$

Introduction to Calculus and Engineering Functions in Matlab

Create Functions in Matlab-

sin(),abs(),..etc are some predefined function now we are going to make a customised function.

```
pramod=inline('abs(x)','x')
```

```
pramod =

     Inline function:
     pramod(x) = abs(x)
```

```
pramod(3)
```

```
ans = 3
```

```
pramod(-5)
```

```
ans = 5
```

```
clear pramod
%pramod(5) ERROR
```

```
new=inline('exp(x)-abs(x)+sqrt(x)','x')
```

```
new =

     Inline function:
     new(x) = exp(x)-abs(x)+sqrt(x)
```

```
new(0)
```

```
ans = 1
```

```
new(-2)
```

```
ans = -1.8647 + 1.4142i
```

```
clear new
```

```
parabolic=inline('sin(x)+cos(y)','x','y')
```

```
parabolic =

     Inline function:
     parabolic(x,y) = sin(x)+cos(y)
```

```
parabolic(2,6)
```

```
ans = 1.8695
```

```
parabolic(0,0)
```

```
ans = 1
```

```
parabolic(pi/2,pi/2)
```

```
ans = 1
```

Anonymous Function

Anonymous Functions :

There are times that we can define a Mathematical function only using one expression instead of a separate function Or separate script. That is when we can use the useful <Anonymous Function>.

We used to have <inline function>, but that is no longer supported in newer version of Matlab. Therefore, we are going To use the <Anonymous function> instead.

F = @ (<arg1>,<arg2>,...) <expression>

Example 1

$f(t) = t^5 e^{-2t} \cos(3t)$

f=@(t) t.^5 .* exp(-2*t) .* cos(3*t)

Example 2

$g = (x, y, a, b, c) = x^a e^{-bx} \cos(cy)$

g= @(x,y,a,b,c) x.^a .* exp(-b.*x) .*cos(c.*y)

```
f=@(t)t.^5.*exp(-2*t).*cos(3*t)
```

```
f = function_handle with value:
    @(t)t.^5.*exp(-2*t).*cos(3*t)
```

```
x=[0:0.1:1]'
```

```
x =
         0
    0.1000
    0.2000
    0.3000
    0.4000
    0.5000
    0.6000
    0.7000
    0.8000
    0.9000
      ⋮
```

```
f(x)
```

```
ans =
         0
    0.0000
    0.0002
    0.0008
    0.0017
    0.0008
   -0.0053
   -0.0209
   -0.0488
   -0.0882
      ⋮
```

```
f(-2)
```

```
ans = -1.6776e+03
```

```
f(0:2)
```

```
ans =
         0   -0.1340    0.5628
```

```
f(2)
```

```
ans = 0.5628
```

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

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

```
f(A)
```

```
ans =
   -0.1340    0.5628   -0.5488
    0.2899   -0.1078    0.0315
   -0.0077    0.0016   -0.0003
```

```
f(rand(6))
```

```
ans =
    0.0001    0.0003   -0.1151    0.0005    0.0004   -0.0013
    0.0000   -0.0336   -0.0012   -0.0538   -0.0071   -0.0959
    0.0009    0.0005    0.0000    0.0004    0.0014    0.0007
   -0.1154    0.0006    0.0000   -0.1013    0.0013   -0.0353
    0.0012   -0.0207    0.0005    0.0013   -0.0599   -0.0343
   -0.0039   -0.0843   -0.0636    0.0002   -0.0039    0.0016
```

```
f=@(t)t.^5*exp(-2*t)*cos(3*t)  %without dot
```

```
f = function_handle with value:
    @(t)t.^5*exp(-2*t)*cos(3*t)
```

```
f(3)
```

```
ans = -0.5488
```

```
%f(-1:0.1:1)  ERROR to perform elementwise multiplication use .*
f=@(t)t.^5.*exp(-2*t).*cos(3*t)
```

```
f = function_handle with value:
    @(t)t.^5.*exp(-2*t).*cos(3*t)
```

```
f([-1:.1:1])
```

```
ans =
    7.3151    3.2296    1.1968    0.3441    0.0587   -0.0060   -0.0083   -0.0028 ...
```

```
g=@cos
```

```
g = function_handle with value:
    @cos
```

```
g(0)
```

```
ans = 1
```

```
g(pi/3)
```

```
ans = 0.5000
```

```
g(pi/2)
```

```
ans = 6.1232e-17
```

```
g=@sin
```

```
g = function_handle with value:
```

```
    @sin
```

```
g(0)
```

```
ans = 0
```

```
g(pi/2)
```

```
ans = 1
```

```
help heaviside
```

```
 heaviside    Step function.
    heaviside(X) is 0 for X < 0 and 1 for X > 0.
    The value heaviside(0) is 0.5 by default. It
    can be changed to any value v by the call
    sympref('HeavisideAtOrigin', v).

    heaviside(X) is not a function in the strict sense.
    See also dirac.

   Reference page for heaviside
   Other functions named heaviside
```

```
t=@(x)((heaviside(x+1)-heaviside(x-1)).*(1-abs(x)))
```

```
t = function_handle with value:
    @(x)((heaviside(x+1)-heaviside(x-1)).*(1-abs(x)))
```

```
t(-1)
```

```
ans = 0
```

```
t(1)
```

```
ans = 0
```

```
t(0)
```

```
ans = 1
```

```
t(2)
```

```
ans = 0
```

```
t(0.5)
```

```
ans = 0.5000
```

```
t(0.3)
```

```
ans = 0.7000
```

```
t(0.75)
```

```
ans = 0.2500
```

```
help ezplot
```

```
 ezplot   (NOT RECOMMENDED) Easy to use function plotter

  ==========================================================
  ezplot is not recommended. Use FPLOT or FIMPLICIT instead.
  ==========================================================

    ezplot(FUN) plots the function FUN(X) over the default domain
    -2*PI < X < 2*PI, where FUN(X) is an explicitly defined function of X.

    ezplot(FUN2) plots the implicitly defined function FUN2(X,Y) = 0 over
    the default domain -2*PI < X < 2*PI and -2*PI < Y < 2*PI.

    ezplot(FUN,[A,B]) plots FUN(X) over A < X < B.
    ezplot(FUN2,[A,B]) plots FUN2(X,Y) = 0 over A < X < B and A < Y < B.

    ezplot(FUN2,[XMIN,XMAX,YMIN,YMAX]) plots FUN2(X,Y) = 0 over
    XMIN < X < XMAX and YMIN < Y < YMAX.

    ezplot(FUNX,FUNY) plots the parametrically defined planar curve FUNX(T)
    and FUNY(T) over the default domain 0 < T < 2*PI.

    ezplot(FUNX,FUNY,[TMIN,TMAX]) plots FUNX(T) and FUNY(T) over
    TMIN < T < TMAX.

    ezplot(FUN,[A,B],FIG), ezplot(FUN2,[XMIN,XMAX,YMIN,YMAX],FIG), or
    ezplot(FUNX,FUNY,[TMIN,TMAX],FIG) plots the function over the
    specified domain in the figure window FIG.

    ezplot(AX,...) plots into AX instead of GCA or FIG.

    H = ezplot(...) returns handles to the plotted objects in H.

    Examples:
    The easiest way to express a function is via a string:
       ezplot('x^2 - 2*x + 1')

    One programming technique is to vectorize the string expression using
    the array operators .* (TIMES), ./ (RDIVIDE), .\ (LDIVIDE), .^ (POWER).
    This makes the algorithm more efficient since it can perform multiple
    function evaluations at once.
       ezplot('x.*y + x.^2 - y.^2 - 1')

    You may also use a function handle to an existing function. Function
    handles are more powerful and efficient than string expressions.
       ezplot(@humps)
       ezplot(@cos,@sin)

    ezplot plots the variables in string expressions alphabetically.
       subplot(1,2,1), ezplot('1./z - log(z) + log(-1+z) + t - 1')
    To avoid this ambiguity, specify the order with an anonymous function:
       subplot(1,2,2), ezplot(@(z,t)1./z - log(z) + log(-1+z) + t - 1)

    If your function has additional parameters, for example k in myfun:
       %-----------------------%
       function z = myfun(x,y,k)
```

```
      z = x.^k - y.^k - 1;
      %-----------------------%
   then you may use an anonymous function to specify that parameter:
      ezplot(@(x,y)myfun(x,y,2))

   See also ezcontour, ezcontourf, ezmesh, ezmeshc, ezplot3, ezpolar,
            ezsurf, ezsurfc, plot, vectorize, function_handle.

   Reference page for ezplot
   Other functions named ezplot
```

```
help fplot
```

```
 fplot   Plot 2-D function
    fplot(FUN) plots the function FUN between the limits of the current
    axes, with a default of [-5 5].

    fplot(FUN,LIMS) plots the function FUN between the x-axis limits
    specified by LIMS = [XMIN XMAX].

    fplot(...,'LineSpec') plots with the given line specification.

    fplot(X,Y,LIMS) plots the parameterized curve with coordinates
    X(T), Y(T) for T between the values specified by LIMS = [TMIN TMAX].

    H = fplot(...) returns a handle to the function line object created by fplot.

    fplot(AX,...) plots into the axes AX instead of the current axes.

    Examples:
        fplot(@sin)
        fplot(@(x) x.^2.*sin(1./x),[-1,1])
        fplot(@(x) sin(1./x), [0 0.1])

    If your function cannot be evaluated for multiple x values at once,
    you will get a warning and somewhat reduced speed:
        f = @(x,n) abs(exp(-1j*x*(0:n-1))*ones(n,1));
        fplot(@(x) f(x,10),[0 2*pi])

    See also fplot3, fsurf, fcontour, fimplicit, plot, function_handle.

    Reference page for fplot
```

```
ezplot(@sin)
fplot(@sin)

ezplot(@tan)
fplot(@tan)
```

```
fplot(@sin,[-1,4])
fplot(@tan,[-2,+2])
```

Introduction to Differentiation

```
syms x
f=inline('sin(x)+2*cos(x)','x')
```

```
f =

     Inline function:
     f(x) = sin(x)+2*cos(x)
```

```
f1=diff(f(x),x)
```

f1 = $\cos(x) - 2\sin(x)$

```
%f1(4) ERROR f1 is not is not recognised as function but just an expression.
% deine f1 in inline function too for f1(4)
derivative=inline(diff(f(x),x),'x')
```

```
derivative =

     Inline function:
     derivative(x) = cos(x)-sin(x).*2.0
```

```
derivative(4)
```

```
ans = 0.8600
```

Differentiation Symbolically

```
syms x
func=x^3
```

func = x^3

```
%func(2) cause error
diff(func,x)
```

ans = $3\,x^2$

```
syms x
g=sin(x)/(x^2+3*cos(x))
```

g =

$$\frac{\sin(x)}{3\cos(x) + x^2}$$

```
diff(g,x)
```

ans =

$$\frac{\cos(x)}{3\cos(x) + x^2} - \frac{\sin(x)\ (2\,x - 3\sin(x))}{(3\cos(x) + x^2)^2}$$

Integration in Matlab

indefinite

```
f1=inline('exp(x)','x')
```

```
f1 =

     Inline function:
     f1(x) = exp(x)
```

```
f1(3)
```

```
ans = 20.0855
```

```
syms x
integral=int(f1(x),x)
```

integral = e^x

```

f1=inline('x','x')
```

```
f1 =

     Inline function:
     f1(x) = x
```

```
integral=int(f1(x),x)
```

integral =

$$\frac{x^2}{2}$$

```
f2=inline('sin(x)+2*cos(x)','x')
```

```
f2 =

     Inline function:
     f2(x) = sin(x)+2*cos(x)
```

```
integral=int(f2(x),x)
```

integral =

$$-2\cos\left(\frac{x}{2}\right)\left(\cos\left(\frac{x}{2}\right)-2\sin\left(\frac{x}{2}\right)\right)$$

```
integral=int(f2(x))
```

integral =

$$-2\cos\left(\frac{x}{2}\right)\ \left(\cos\left(\frac{x}{2}\right)-2\sin\left(\frac{x}{2}\right)\right)$$

```
f3=inline('x*y+sin(x+y)-tan(y)+3*exp(x^3)','x','y')
```

```
f3 =

     Inline function:
     f3(x,y) = x*y+sin(x+y)-tan(y)+3*exp(x^3)
```

```
syms y
integral=int(f3(x,y),x)
```

```
integral =
```

$$\frac{x^2\,y}{2}-x\,\mathrm{expint}\left(\frac{2}{3},-x^3\right)-\cos(x+y)-x\tan(y)$$

```
inegral=int(f3(x,y),y)
```

```
inegral =
```

$$3\,y\,e^{x^3}-\cos(x+y)-\frac{\log(\tan(y)^2+1)}{2}+\frac{x\,y^2}{2}$$

definite

```
f1
```

```
f1 =

     Inline function:
     f1(x) = x
```

```
intgral=int(f1(x),-12,.8)
```

```
intgral =
```

$$-\frac{1792}{25}$$

```
f2
```

```
f2 =

     Inline function:
     f2(x) = sin(x)+2*cos(x)
```

```
integral=int(f2(x),5,-5)
```

integral = $-4\sin(5)$

```
double(ans)
```

ans = 20.0855

```
f3
```

f3 =

    Inline function:
    f3(x,y) = x*y+sin(x+y)-tan(y)+3*exp(x^3)

```
integral=int(f3(x,y),x)
```

integral =

$$\frac{x^2\,y}{2} - x\,\mathrm{expint}\left(\frac{2}{3}, -x^3\right) - \cos(x+y) - x\tan(y)$$

```
integral=int(f3(x,y),-1,4)
```

integral =

$$\frac{15\,y}{2} + \cos(y-1) - \cos(y+4) - 5\tan(y) - \mathrm{expint}\left(\frac{2}{3}, 1\right) - 4\,\mathrm{expint}\left(\frac{2}{3}, -64\right) + 3\left(\lim_{x\to 0^-} \sigma_1\right) - 3\left(\lim_{x\to 0^+} \sigma_1\right)$$

where

$$\sigma_1 = -\frac{x\,\mathrm{expint}\left(\frac{2}{3}, -x^3\right)}{3}$$

LImit function

```
f1=inline('sin(x)/x','x')
```

f1 =

    Inline function:
    f1(x) = sin(x)/x

```
syms x
limit(f1(x),x,0)
```

ans = 1

```
syms x y
f2=inline('(sin(x+y)-sin(x)/y)','x','y')
```

```
f2 =

     Inline function:
     f2(x,y) = (sin(x+y)-sin(x)/y)
```

```
limit(f2(x,y),x,0)
```

ans = $\sin(y)$

```
h=inline((4*x^3-x^2+2*x-1)/(3*x^3-7*x^2-1),'x')
```

```
h =

     Inline function:
     h(x) = -(x.*2.0-x.^2+x.^3.*4.0-1.0)./(x.^2.*7.0-x.^3.*3.0+1.0)
```

```
syms x
pretty(h(x))
```

```
      3    2
   4 x  - x  + 2 x - 1
- -------------------
         3      2
   - 3 x  + 7 x  + 1
```

```
limit(h(x),x,0)
```

ans = 1

```
limit(h(x),x,inf)
```

ans =

$$\frac{4}{3}$$

```
syms x a
v=[(1+a/x)^x exp(-x)]
```

v =

$$\left(\left(\frac{a}{x}+1\right)^x \quad \mathrm{e}^{-x}\right)$$

```
limit(v,x,inf)
```

ans = $\left(\mathrm{e}^a \quad 0\right)$

Partial Derivatives

```
syms x y
f1=inline('x^2+2*y^3','x','y')
```

```
f1 =

     Inline function:
     f1(x,y) = x^2+2*y^3
```

```
d1=diff(f1(x,y),x)
```

d1 = $2\,x$

```
d2=diff(f1(x,y),y)
```

d2 = $6\,y^2$

```

f2=inline('x*sin(y)-y^3*sqrt(x^4)','x','y')
```

```
f2 =

     Inline function:
     f2(x,y) = x*sin(y)-y^3*sqrt(x^4)
```

```
d1=diff(f2(x,y),x)
```

d1 =

$$\sin(y) - \frac{2\,x^3\,y^3}{\sqrt{x^4}}$$

```
d2=diff(f2(x,y),y)
```

d2 = $x\cos(y) - 3\,y^2\,\sqrt{x^4}$

```

%d2(1,4) ERROR not defined as function for matlab
g2=inline(diff(f2(x,y),y),'x','y')
```

```
g2 =

     Inline function:
     g2(x,y) = y.^2.*sqrt(x.^4).*-3.0+x.*cos(y)
```

```
g2(1,2)
```

```
ans = -12.4161
```

```
g2(-1,37)
```

```
ans = -4.1078e+03
```

Graphs & Plots in Matlab

```
x=[1 2 3 4 5 6 7 8 9]
```

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

```
y=[-1 3 -2 7 8 9 3 6 9]
```

```
y =
    -1     3    -2     7     8     9     3     6     9
```

```
plot(x,y)
xlabel('x values')
ylabel('y values')
title('my first graph')
grid on
```

```
x
```

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

```
y
```

```
y =
    -1     3    -2     7     8     9     3     6     9
```

```
plot(x,y,'r')
plot(x,y,'--m*')
plot(x,y,'-.+')
plot(x,y,'-.bo')
plot(x,y,'^')
plot(x,y,'s')
```

line specification in single ' '

```
x=[1 2 3 4 5 6 7 8 9]
```

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

```
y=[-1 3 -2 7 8 9 3 6 9]
```

```
y =
```

```
    -1     3    -2     7     8     9     3     6     9
```

```
plot(x,y,'om-.')
```

Plotting more than one function on single plot - hold on/off

```
x=0:pi/150:2*pi
```

```
x =
         0    0.0209    0.0419    0.0628    0.0838    0.1047    0.1257    0.1466 ···
```

```
y1=sin(4*x)
```

```
y1 =
         0    0.0837    0.1668    0.2487    0.3289    0.4067    0.4818    0.5534 ···
```

```
plot(x,y1)
hold on
y2=sin(x)
```

```
y2 =
         0    0.0209    0.0419    0.0628    0.0837    0.1045    0.1253    0.1461 ···
```

```
plot(x,y2,'r')
```

```
plot(x,y1,'r+--',x,y2,'c:d')
```

```
hold off
y1=sin(x)*2
```

```
y1 =
         0    0.0419    0.0838    0.1256    0.1674    0.2091    0.2507    0.2922 ···
```

```
y2=sin(x)*3
```

```
y2 =
         0    0.0628    0.1256    0.1884    0.2510    0.3136    0.3760    0.4382 ···
```

```
y3=sin(x)*4
```

```
y3 =
         0    0.0838    0.1675    0.2512    0.3347    0.4181    0.5013    0.5843 ···
```

```
y4=sin(x)*5
```

```
y4 =
         0    0.1047    0.2094    0.3140    0.4184    0.5226    0.6267    0.7304 ···
```

```
plot(x,y1,'o',x,y2,'--',x,y3,'d',x,y4,'s')
```

Subplot

subplot will have three element- subplot(m,n,p)

```
help subplot
```

```
subplot Create axes in tiled positions.
   H = subplot(m,n,p), or subplot(mnp), breaks the Figure window
   into an m-by-n matrix of small axes, selects the p-th axes for
   the current plot, and returns the axes handle.  The axes are
   counted along the top row of the Figure window, then the second
   row, etc.  For example,

       subplot(2,1,1), PLOT(income)
       subplot(2,1,2), PLOT(outgo)

   plots income on the top half of the window and outgo on the
   bottom half. If the CurrentAxes is nested in a uipanel the
   panel is used as the parent for the subplot instead of the
   current figure.

   subplot(m,n,p), if the axes already exists, makes it current.
   subplot(m,n,p,'replace'), if the axes already exists, deletes it and
   creates a new axes.
   subplot(m,n,p,'align') places the axes so that the plot boxes
   are aligned, but does not prevent the labels and ticks from
   overlapping.
   subplot(m,n,P), where P is a vector, specifies an axes position
   that covers all the subplot positions listed in P.
   subplot(H), where H is an axes handle, is another way of making
   an axes current for subsequent plotting commands.

   subplot('position',[left bottom width height]) creates an
   axes at the specified position in normalized coordinates (in
   in the range from 0.0 to 1.0).

   subplot(..., PROP1, VALUE1, PROP2, VALUE2, ...) sets the
   specified property-value pairs on the subplot axes. To add the
   subplot to a specific figure pass the figure handle as the
   value for the 'Parent' property.

   If a subplot specification causes a new axes to overlap an
   existing axes, the existing axes is deleted - unless the position
   of the new and existing axes are identical.  For example,
   the statement subplot(1,2,1) deletes all existing axes overlapping
   the left side of the Figure window and creates a new axes on that
   side - unless there is an axes there with a position that exactly
   matches the position of the new axes (and 'replace' was not specified),
   in which case all other overlapping axes will be deleted and the
   matching axes will become the current axes.

   subplot(111) is an exception to the rules above, and is not
   identical in behavior to subplot(1,1,1).  For reasons of backwards
   compatibility, it is a special case of subplot which does not
   immediately create an axes, but instead sets up the figure so that
   the next graphics command executes CLF RESET in the figure
   (deleting all children of the figure), and creates a new axes in
   the default position.  This syntax does not return a handle, so it
   is an error to specify a return argument.  The delayed CLF RESET
   is accomplished by setting the figure's NextPlot to 'replace'.

   Be aware when creating subplots from scripts that the Position
```

```
    property of subplots is not finalized until either a drawnow
    command is issued, or MATLAB returns to await a user command.
    That is, the value obtained for subplot i by the command
    h(i).Position will not be correct until the script
    refreshes the plot or exits.

    See also  gca, gcf, axes, figure, uipanel

    Reference page for subplot
```

```
subplot(2,2,1)
plot(x,sin(x))

subplot(2,1,1)
plot(x,sin(x))

subplot(2,1,2)
plot(x,cos(x))
plot(x,cos(x),'r--')
```

```
y1=sin(x)*2;
y2=sin(x)*3;
y3=sin(x)*4;
y4=sin(x)*5;
bar3(y1)
```

```
x=[1 2 5 3 9];
y=[x;1:5]
```

```
y =
     1     2     5     3     9
     1     2     3     4     5
```

```
subplot(2,2,1)
bar(x),title('a bar graph for vector x')
subplot(2,2,2)
bar(y),title('bar graph for y')
subplot(2,2,3)
bar3(y),title('this is a bar3 graph for y ')
subplot(2,2,4)
pie(x),title('this is apie graph for x')
clear
```

plot in easy way

```
close all hidden
x=0:0.5:2*pi
```

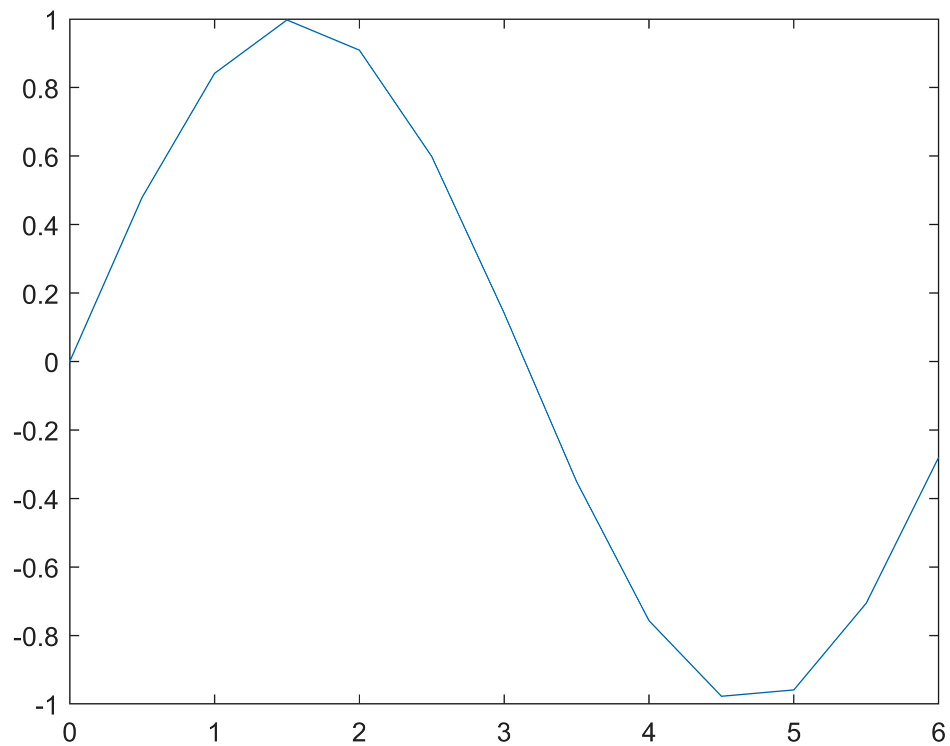
```
x =
```

```
         0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000 ⋯
```

```
y=sin(x)
```

```
y =
         0    0.4794    0.8415    0.9975    0.9093    0.5985    0.1411   -0.3508 ⋯
```

```
plot(x,y)
```



Loops, Conditions, and Intro to Programming in Matlab

```
x=4;
y=-2;
x<y
```

```
ans = logical
   0
```

```
x>y
```

```
ans = logical
   1
```

```
x=1:5
```

```
x =
     1     2     3     4     5
```

```
y=x+3
```

```
y =
     4     5     6     7     8
```

```
x<y
```

```
ans = 1×5 logical array
   1   1   1   1   1
```

```
x>y
```

```
ans = 1×5 logical array
   0   0   0   0   0
```

```
y=[2 1 -1 5 9]
```

```
y =
     2     1    -1     5     9
```

```
x>y
```

```
ans = 1×5 logical array
   0   1   1   0   0
```

```
y>x
```

```
ans = 1×5 logical array
   1   0   0   1   1
```

```
x<=y
```

```
ans = 1×5 logical array
   1   0   0   1   1
```

```
x>=y
```

```
ans = 1×5 logical array
   0   1   1   0   0
```

```
x==y
```

```
ans = 1×5 logical array
   0   0   0   0   0
```

```
x~=y %not equal
```

```
ans = 1×5 logical array
   1   1   1   1   1
```

```
x=[-1 2 0 4 6]
```

```
x =
    -1     2     0     4     6
```

```
y=[2 1 0 5 9]
```

```
y =
     2     1     0     5     9
```

```
x~=y
```

```
ans = 1×5 logical array
   1   1   0   1   1
```

```
x==y
```

```
ans = 1×5 logical array
   0   0   1   0   0
```

Logical operator

and &

not -

or |

exclusively or xor

```
x=[1 2 3 4 5];
y=[-2 0 2 4 6];
x<y
```

```
ans = 1×5 logical array
   0   0   0   0   1
```

```
z=[8 8 8 8 8];
z>x & z>y
```

```
ans = 1×5 logical array
   1   1   1   1   1
```

```
z>x | z>y
```

```
ans = 1×5 logical array
   1   1   1   1   1
```

```
x>y | x>z
```

```
ans = 1×5 logical array
   1   1   1   0   0
```

```
x>y & x>z
```

```
ans = 1×5 logical array
   0   0   0   0   0
```

```
speed=[63 67 65 55 69 40 95]
```

```
speed =
    63    67    65    55    69    40    95
```

```
result=find(speed>=65)
```

```
result =
     2     3     5     7
```

```
speed(result)
```

```
ans =
    67    65    69    95
```

Conditions in Matlab and and Introduction to Else

```
x=3;
if x<4
    disp('x is les than 4')
    disp(x)
end
```

```
x is les than 4
     3
```

```
x=5;
if x<4
    disp('x is les than 4')
    disp(x)
end
```

```
x=5
```

```
x = 5
```

```
if x>0
    y=log(x)
else
    disp("input of log can't be negative")
end
```

```
y = 1.6094
```

```
x=-6
```

```
x = -6
```

```
if x>0
    y=log(x)
else
    disp("input of log can't be negative")
end
```

```
input of log can't be negative
```

for loop

```
x=-4
```

```
x = -4
```

```
if x>0
    y=log(x)
else
    beep
    disp("input of log can't be negative")
end
```

```
input of log can't be negative
```

```
scores=[30 65 91 87 56 93 52 99 90];
count=0
```

```
count = 0
```

```
for k=1:length(scores)
    if scores(k)>90
        count=count+1
    end
end
```

```
count = 1
count = 2
count = 3
```

```
disp(count)
```

```
     3
```

```
scores=[30 65 91 87 56 93 52 99 90];
count=0;
for k=1:length(scores)
    if scores(k)>90
        count=count+1;
    end
end
disp(count)
```

```
     3
```

elseif

```
age=15
```

```
age = 15
```

```
if age<16
    disp('sorry you are too young too apply')
elseif age<18
    disp('yopu can apply for youth lecense')
elseif age<70
    disp('you may have standerd driving license')
else
    disp('driver over 70 will require aspecial license')
end
```

```
sorry you are too young too apply
```

```
age=16
```

```
age = 16
```

```
if age<16
    disp('sorry you are too young too apply')
elseif age<18
    disp('yopu can apply for youth lecense')
elseif age<70
    disp('you may have standerd driving license')
else
```

```
    disp('driver over 70 will require aspecial license')
end
```

```
yopu can apply for youth lecense
```

```
age=65;
if age<16
    disp('sorry you are too young too apply')
elseif age<18
    disp('yopu can apply for youth lecense')
elseif age<70
    disp('you may have standerd driving license')
else
    disp('driver over 70 will require aspecial license')
end
```

```
you may have standerd driving license
```

```
age=71;
if age<16
    disp('sorry you are too young too apply')
elseif age<18
    disp('yopu can apply for youth lecense')
elseif age<70
    disp('you may have standerd driving license')
else
    disp('driver over 70 will require aspecial license')
end
```

```
driver over 70 will require aspecial license
```

while loop

```
k=0;
while k<3
    k=k+1
end
```

```
k = 1
k = 2
k = 3
```

```
scores=[30 65 91 87 56 93 52 99 90];
count=0;
k=0;
while k<length(scores)
    k=k+1;
```

```
    if scores(k)>90
        count=count+1;

    end
end
disp(count)
```

```
     3
```

for index[matrix]

commands to be execute

end

```
for i=[11 52 83]
    i
end
```

```
i = 11
i = 52
i = 83
```

```
i
```

```
i = 83
```

```
for k=1:4
    a=6^k
end
```

```
a = 6
a = 36
a = 216
a = 1296
```

Let's Code some Matlab Projects in Matlab

```
x=[12 13 87 31]
```

```
x =
    12    13    87    31
```

```
N=[5 1 7 54]
```

```
N =
```

```
     5     1     7    54
```

```
xsize= size(x)
```

```
xsize =
     1     4
```

```
Nsize= size(N)
```

```
Nsize =
     1     4
```

```
if Nsize(2)~=xsize(2)
    disp('Error-size should be equa')
else
    total=sum(N)
    average=sum(x.*N)/total
end
```

```
total = 67
average = 35.1642
```

Project 1- Let's Create a function in Matlab with Example

```
%or we can save a function
%here first argument in Numbers, and second is repitation.
% function average=avg(x,N)
%     xsize= size(x)
%     Nsize= size(N)
%     if Nsize(2)~=xsize(2)
%         disp('Error-size should be equa')
%     else
%         total=sum(N)
%         average=sum(x.*N)/total
%     end
% end

avg(x,N)
```

```
average = 35.1642
ans = 35.1642
```

```
y=[1 25 6 68 97 85 24 56]
```

```
y =
     1    25     6    68    97    85    24    56
```

```
M=[1 4 5 8 6 3 25 98]
```

```
M =
     1     4     5     8     6     3    25    98
```

```
avg(y,M)
```

```
average = 50.6667
ans = 50.6667
```

```
avg(y,N)
```

```
Error-size should be equa
```

```
avg(x,M)
```

```
Error-size should be equa
```

Project 2: How to evaluate a Polynomial for any number in Matlab

```
% function output=poly(x)
% output=458.*x.^3+2.5*x.^2+137.*x+1;
% end

poly(1)
```

```
ans = 621
```

```
poly(17)
```

```
ans = 2259709
```

```
poly([1 2 3 45])
```

```
ans =
         621        4039       13003    41792041
```

Project 3 : Automate finding the area of a triangle for any base and height

```
a = input('enter the value for a: ')
```

```
a = 10
```

```
name=input('what is your name: ') %here string are not acceptable
```

```
name = 20
```

```
%for string input
```

```
name=input('what is your name: ','s')
```

```
name =
'pky'
```

num2str

```
x=[1 2 3]
```

```
x =
     1     2     3
```

```
y=[5 9 8]
```

```
y =
     5     9     8
```

```
x+y
```

```
ans =
     6    11    11
```

```
x=num2str(x)
```

```
x =
'1  2  3'
```

```
%x+y  here this error as x ix now a string
```

```
%in this we are developing program to find area
% of triange via side input by user

b=input('enter the value of base: ');
h=input('enter the value oh height: ');
triangle_area=0.5*h*b;
disp(['the value for area is: ' num2str(triangle_area)])
```

```
the value for area is: 130
```

Import Excel in Matlab and change the data

- Put your ecxel file in matlab directory
- XLSREAD('name of the file')
- diffrent argument when importing excel file into MATLAB- **[num,text,raw]=xlsread('name_of_the_xl_file')**
- 'ls' command give all xl file in the directory

```
ls
```

```
.                         MATLAB_Introduction1.pdf  poly.m
..                        anonymous.png             test_book.xlsx
MATLAB_Introduction1.mlx  avg.m
```

```
data=xlsread('test_book')
```

```
data =
    1.0000   52.0000  160.0000    2.0000    0.1783    0.0857    2.0000
    2.0000   49.0000  150.0000    3.0000    0.7597    0.2586    2.0000
    3.0000   39.0000  122.0000    1.0000    0.9149    0.3568    2.0000
    4.0000   58.0000  133.0000    5.0000    0.8159    0.2589    3.0000
    5.0000   52.0000  148.0000    3.0000    0.9169    0.7852    3.0000
    6.0000   41.0000  147.0000    2.0000    0.6169    0.1785    3.0000
    7.0000   32.0000  158.0000    1.0000    0.9179    0.6785    5.0000
    8.0000   65.0000  169.0000    2.0000    0.8199    0.5785    2.0000
    9.0000   45.0000   45.0000    3.0000    0.7149    0.6785    2.0000
   10.0000   52.0000  163.0000    4.0000    0.9186    0.7852    2.0000
      ⋮
```

```
[A,B,C]=xlsread('test_book')
```

```
A =
    1.0000   52.0000  160.0000    2.0000    0.1783    0.0857    2.0000
    2.0000   49.0000  150.0000    3.0000    0.7597    0.2586    2.0000
    3.0000   39.0000  122.0000    1.0000    0.9149    0.3568    2.0000
    4.0000   58.0000  133.0000    5.0000    0.8159    0.2589    3.0000
    5.0000   52.0000  148.0000    3.0000    0.9169    0.7852    3.0000
    6.0000   41.0000  147.0000    2.0000    0.6169    0.1785    3.0000
    7.0000   32.0000  158.0000    1.0000    0.9179    0.6785    5.0000
    8.0000   65.0000  169.0000    2.0000    0.8199    0.5785    2.0000
    9.0000   45.0000   45.0000    3.0000    0.7149    0.6785    2.0000
   10.0000   52.0000  163.0000    4.0000    0.9186    0.7852    2.0000
      ⋮

B = 1×7 cell array
    {'time'}    {'pressure'}    {'temp'}    {'level'}    {'density'}    {'viscocity'}    {'velocity'}
C = 19×7 cell array
    {'time'}    {'pressure'}    {'temp'}    {'level'}    {'density'}    {'viscocity'}    {'velocity'}
    {[   1]}    {[        52]}  {[ 160]}    {[     2]}   {[ 0.1783]}    {[   0.0857]}    {[        2]}
    {[   2]}    {[        49]}  {[ 150]}    {[     3]}   {[ 0.7597]}    {[   0.2586]}    {[        2]}
    {[   3]}    {[        39]}  {[ 122]}    {[     1]}   {[ 0.9149]}    {[   0.3568]}    {[        2]}
    {[   4]}    {[        58]}  {[ 133]}    {[     5]}   {[ 0.8159]}    {[   0.2589]}    {[        3]}
    {[   5]}    {[        52]}  {[ 148]}    {[     3]}   {[ 0.9169]}    {[   0.7852]}    {[        3]}
    {[   6]}    {[        41]}  {[ 147]}    {[     2]}   {[ 0.6169]}    {[   0.1785]}    {[        3]}
    {[   7]}    {[        32]}  {[ 158]}    {[     1]}   {[ 0.9179]}    {[   0.6785]}    {[        5]}
    {[   8]}    {[        65]}  {[ 169]}    {[     2]}   {[ 0.8199]}    {[   0.5785]}    {[        2]}
    {[   9]}    {[        45]}  {[  45]}    {[     3]}   {[ 0.7149]}    {[   0.6785]}    {[        2]}
    {[  10]}    {[        52]}  {[ 163]}    {[     4]}   {[ 0.9186]}    {[   0.7852]}    {[        2]}
    {[  11]}    {[        55]}  {[ 123]}    {[     5]}   {[ 0.8561]}    {[   0.5785]}    {[        2]}
    {[  12]}    {[        49]}  {[ 157]}    {[     2]}   {[ 0.9566]}    {[   0.8785]}    {[        3]}
    {[  13]}    {[        48]}  {[ 145]}    {[     3]}   {[ 0.9149]}    {[   0.9785]}    {[        9]}
    {[  14]}    {[        45]}  {[  56]}    {[     4]}   {[ 0.9179]}    {[   0.4785]}    {[        5]}
    {[  15]}    {[        42]}  {[   6]}    {[     2]}   {[ 0.8157]}    {[   0.5785]}    {[        2]}
    {[  16]}    {[        47]}  {[ 125]}    {[     2]}   {[ 0.8210]}    {[   0.6785]}    {[        4]}
```

```
    {[  17]}    {[       35]}    {[ 153]}    {[     2]}    {[ 0.8156]}    {[    0.7785]}    {[         3]}
    {[  18]}    {[       44]}    {[ 158]}    {[     3]}    {[ 0.9358]}    {[    0.6785]}    {[         4]}
```

```
[~,B,~]=xlsread('test_book')
```

```
B = 1×7 cell array
    {'time'}    {'pressure'}    {'temp'}    {'level'}    {'density'}    {'viscocity'}    {'velocity'}
```

```
[A,~,~]=xlsread('test_book')
```

```
A =
    1.0000   52.0000  160.0000    2.0000    0.1783    0.0857    2.0000
    2.0000   49.0000  150.0000    3.0000    0.7597    0.2586    2.0000
    3.0000   39.0000  122.0000    1.0000    0.9149    0.3568    2.0000
    4.0000   58.0000  133.0000    5.0000    0.8159    0.2589    3.0000
    5.0000   52.0000  148.0000    3.0000    0.9169    0.7852    3.0000
    6.0000   41.0000  147.0000    2.0000    0.6169    0.1785    3.0000
    7.0000   32.0000  158.0000    1.0000    0.9179    0.6785    5.0000
    8.0000   65.0000  169.0000    2.0000    0.8199    0.5785    2.0000
    9.0000   45.0000   45.0000    3.0000    0.7149    0.6785    2.0000
   10.0000   52.0000  163.0000    4.0000    0.9186    0.7852    2.0000
      ⋮
```

```
[~,~,C]=xlsread('test_book')
```

```
C = 19×7 cell array
    {'time'}    {'pressure'}    {'temp'}    {'level'}    {'density'}    {'viscocity'}    {'velocity'}
    {[   1]}    {[       52]}    {[ 160]}    {[     2]}    {[ 0.1783]}    {[    0.0857]}    {[         2]}
    {[   2]}    {[       49]}    {[ 150]}    {[     3]}    {[ 0.7597]}    {[    0.2586]}    {[         2]}
    {[   3]}    {[       39]}    {[ 122]}    {[     1]}    {[ 0.9149]}    {[    0.3568]}    {[         2]}
    {[   4]}    {[       58]}    {[ 133]}    {[     5]}    {[ 0.8159]}    {[    0.2589]}    {[         3]}
    {[   5]}    {[       52]}    {[ 148]}    {[     3]}    {[ 0.9169]}    {[    0.7852]}    {[         3]}
    {[   6]}    {[       41]}    {[ 147]}    {[     2]}    {[ 0.6169]}    {[    0.1785]}    {[         3]}
    {[   7]}    {[       32]}    {[ 158]}    {[     1]}    {[ 0.9179]}    {[    0.6785]}    {[         5]}
    {[   8]}    {[       65]}    {[ 169]}    {[     2]}    {[ 0.8199]}    {[    0.5785]}    {[         2]}
    {[   9]}    {[       45]}    {[  45]}    {[     3]}    {[ 0.7149]}    {[    0.6785]}    {[         2]}
    {[  10]}    {[       52]}    {[ 163]}    {[     4]}    {[ 0.9186]}    {[    0.7852]}    {[         2]}
    {[  11]}    {[       55]}    {[ 123]}    {[     5]}    {[ 0.8561]}    {[    0.5785]}    {[         2]}
    {[  12]}    {[       49]}    {[ 157]}    {[     2]}    {[ 0.9566]}    {[    0.8785]}    {[         3]}
    {[  13]}    {[       48]}    {[ 145]}    {[     3]}    {[ 0.9149]}    {[    0.9785]}    {[         9]}
    {[  14]}    {[       45]}    {[  56]}    {[     4]}    {[ 0.9179]}    {[    0.4785]}    {[         5]}
    {[  15]}    {[       42]}    {[   6]}    {[     2]}    {[ 0.8157]}    {[    0.5785]}    {[         2]}
    {[  16]}    {[       47]}    {[ 125]}    {[     2]}    {[ 0.8210]}    {[    0.6785]}    {[         4]}
    {[  17]}    {[       35]}    {[ 153]}    {[     2]}    {[ 0.8156]}    {[    0.7785]}    {[         3]}
    {[  18]}    {[       44]}    {[ 158]}    {[     3]}    {[ 0.9358]}    {[    0.6785]}    {[         4]}
```

```
presure=A(1:end,2)
```

```
presure = 18×1
    52
    49
    39
```

58
52
41
32
65
45
52
⋮

*** **END** ***