AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

UTTAR PRADESH

ASET
Department of Computer Science and Engineering

BASIC SIMULATION LAB

PRACTICAL FILE

Submitted to-
Dr. Rinki Gupta

Submitted by-
Sanya Sachdeva A2305219131
3CSE-2Y
Btech 2019-23

INDEX

5	Generating a set of Commands on a given Vector to- a. Create a vector Z with elements, $Z = \frac{(-1)^{n+1}}{2n-1}$ Add up to 100 elements of the vector Z. Plot Z b. Plot the functions, $x, x^3, e^x, \exp(x^2)$ over the interval $0<x<4$ (by choosing appropriate mesh values for x to obtain smooth curves), on a rectangular plot.	34	Aug 26	Aug 26	Aug 26	1
6	Generating a set of Commands on a given Vector to- a. Generating a Sinusoidal Signal of a given frequency with Titling, Labelling, Adding Text, Adding Legends, Printing Text in Greek Letters, Plotting as Multiple and Subplot. b. Time scale the generated signal for different values. E.g. 2X, 4X, 0.25X, 0.0625X.	39	Sep 2	Sep 2	Sep 9	1
7	Solving Ordinary Differential Equation using Built-in Functions and plot. a. First Order ordinary differential equation b. Second Order ordinary differential equation c. Third Order ordinary differential equation	44	Sep 9	Sep 9	Sep 23	1
8	Writing brief Scripts starting each Script with a request for input (using input) to Evaluate the function h(T) using if-else statement, where, $h(T) = (T – 10)$ for $0 < T < 100$ $h(T) = (0.45 T + 900)$ for $T > 100$.	47	Sep 23	Sep 23	Oct 14	1
9	Generating a Square Wave from sum of Sine Waves of certain Amplitude and Frequencies.	52	Sep 30	Sep 30	Oct 14	1
10	Basic 2D and 3D plots: parametric space curve, polygons with vertices, 3D contour lines and pie and bar charts.	57	Oct 7	Oct 7	Oct 14	1
						10

Experiment: 1

Aim:

Creating a One and Two-Dimensional Array (Row / Column Vector) (Matrix of given size) then,

(A). Performing Arithmetic Operations - Addition, Subtraction, Multiplication and Exponentiation.

(B). Performing Matrix operations - Inverse, Transpose, Rank with plots.

Description:

MATLAB is an abbreviation for "matrix laboratory." All MATLAB variables are multidimensional arrays, no matter what type of data. A *matrix* is a two-dimensional array often used for linear algebra.

I. Array Creation:

   To create an array with elements in a single row, separate elements with either a comma (,) or a space and to add multiple rows use semicolon(;).

II. Arithmetic operations:

   a. Addition: +
   b. Subtraction: -
   c. Multiplication: *
   d. Element-wise multiplication: **.***
   e. Exponentiation:

      Y = exp(X) ,which returns the exponential e^x for each element in array X.

III. Matrix operations:

   a. Inverse:

      Y = inv(X), which computes the inverse of square matrix X.

   b. Transpose:

      B = A.'

      B = transpose(A) , which returns the nonconjugate transpose of A, that is, interchanges the row and column index for each element.

   c. Rank with plots:

      k = rank(A) , which returns the rank of matrix A.

Code:

1. Array and matrix creation:

```
a=1 %variable
b=[1 2 3;4 5 6;7 8 9] %matrix
d=[1 2 3;4 1 6;1 8 9]
g=[1 2 3;4 5 6]
f=[1 2;3 4;5 6]
%matrix
e= [10 20 30; 40 10 60;10 80
    90]
```

2. Arithmetic operations:

```
%arithmetic operations
10+20
30*5
4/2
39-9.6
```

3. Arithmetic operation on vectors:

```
%arithmetic operations on
    vectors
r1=b+e
r2=b-e
r3=g*f
r4=b/d
a=[1 2 3]
b=[1;2;3]
```

4. Arithmetic operation on matrix:

```
%arithmetic operations on
    matrix
r4= d.*e
r1a=d+e
r2a=d-e
r3a=d.*e
r4a=d./e
```

5. Exponentiation:

```
%exponential
r5=exp(e)
```

6. Inverse :

```
#inverse
inverse_e=inv(e)

%re inverse
inverse2_e=inv(inv(e))
```

7. Transverse :

```
%transverse
disp(e)
trans_e=e'
```

8. Rank:

```
%rank
rank_d=rank(e)
```

9. Rank with plots:

```
%plot
%using column by column only
figure;plot(a)
figure;plot(b)
figure;plot(d)
figure;plot(g)
figure;plot(f)
```

Results:

1. Total variables:

Vars
[1x3] a
ans
[3x1] b
[3x3] d
[3x3] e
[3x2] f
[2x3] g
[3x3] inverse2_e
[3x3] inverse_e
[3x3] r1
[3x3] r1a
[3x3] r2
[3x3] r2a
[2x2] r3
[3x3] r3a
[3x3] r4
[3x3] r4a
[3x3] r5
rank_d
[3x3] trans_e

2. Array and matrix creation:

```
a = 1
b =

   1   2   3
   4   5   6
   7   8   9

d =

   1   2   3
   4   1   6
   1   8   9

g =

   1   2   3
   4   5   6
```

```
f =

   1   2
   3   4
   5   6

e =

   10   20   30
   40   10   60
   10   80   90
```

```
a1 =

   1   2   3

b1 =

   1
   2
   3
```

3. Arithmetic operations:

```
ans = 30
ans = 150
ans = 2
ans = 29.400
```

4. Arithmetic operation on vectors:

```
r1 =

   11   22   33
   44   15   66
   17   88   99

r2 =

   -9  -18  -27
  -36   -5  -54
   -3  -72  -81
```

```
r3 =

   22   28
   49   64

r4 =

    1.0000         0         0
   20.0000   -3.0000   -4.0000
   39.0000   -6.0000   -8.0000
```

5. Arithmetic operation on matrix:

```
r41 =

    10    40    90
   160    10   360
    10   640   810

r1a =

   11   22   33
   44   11   66
   11   88   99

r2a =

   -9  -18  -27
  -36   -9  -54
   -9  -72  -81
```

```
r3a =

    10    40    90
   160    10   360
    10   640   810

r4a =

   0.1000   0.1000   0.1000
   0.1000   0.1000   0.1000
   0.1000   0.1000   0.1000
```

6. Exponentiation:

```
r5 =

   2.2026e+04   4.8517e+08   1.0686e+13
   2.3539e+17   2.2026e+04   1.1420e+26
   2.2026e+04   5.5406e+34   1.2204e+39
```

7. Inverse :

```
inverse_e =

   0.6500  -0.1000  -0.1500
   0.5000  -0.1000  -0.1000
  -0.5167   0.1000   0.1167

inverse2_e =

   10.0000   20.0000   30.0000
   40.0000   10.0000   60.0000
   10.0000   80.0000   90.0000
```

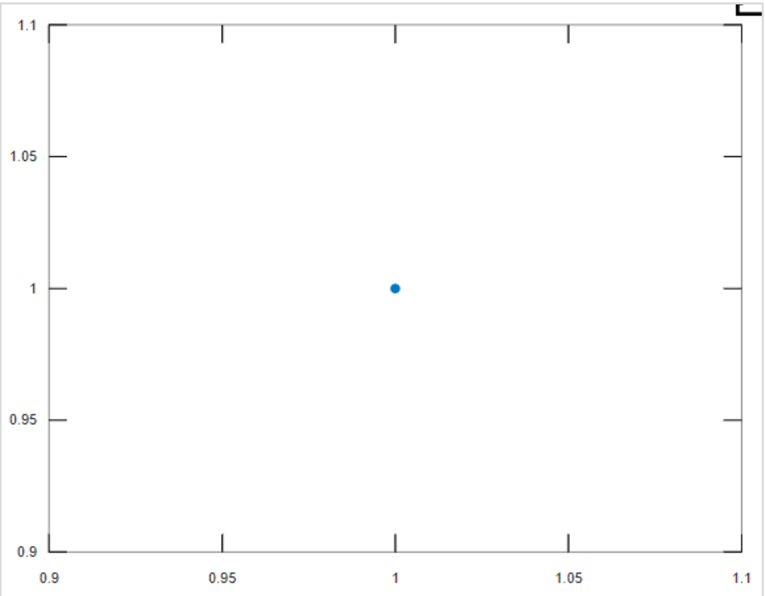
8. Transverse :

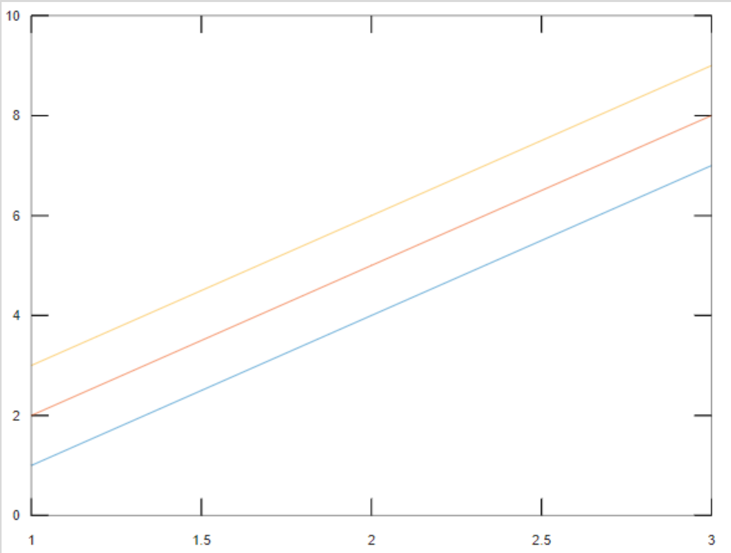
```
trans_e =

   10   40   10
   20   10   80
   30   60   90
```

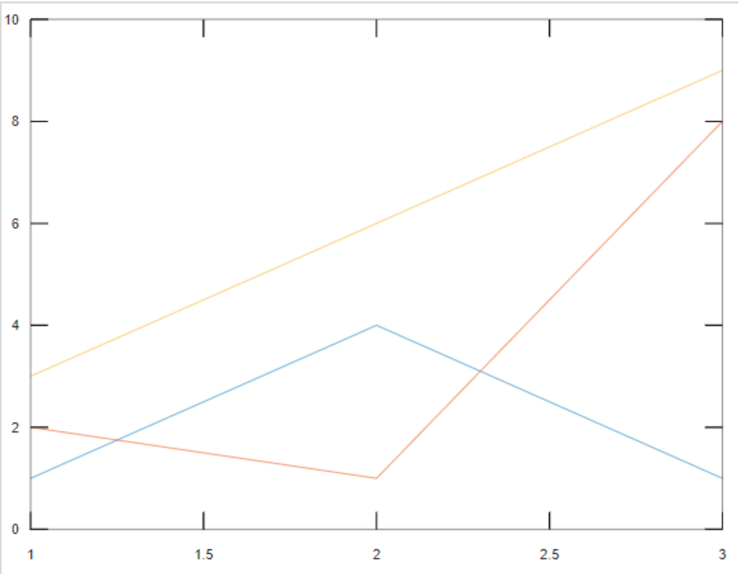
9. Rank:

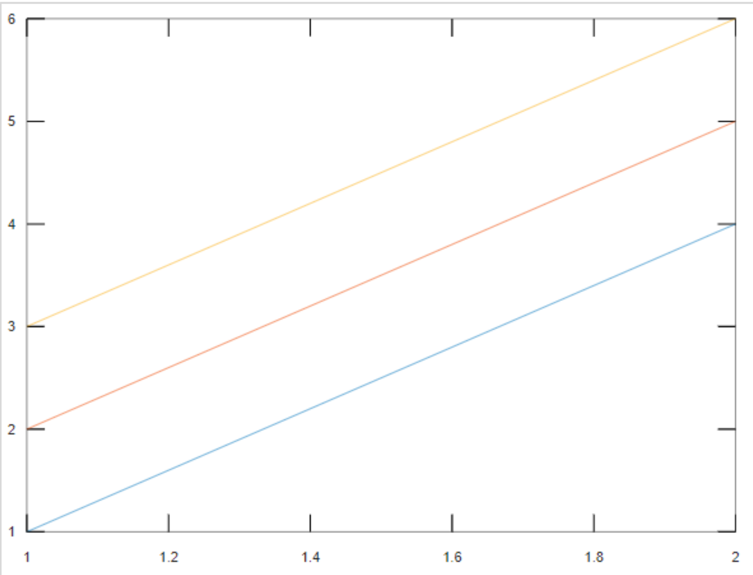
```
rank_d = 3
```

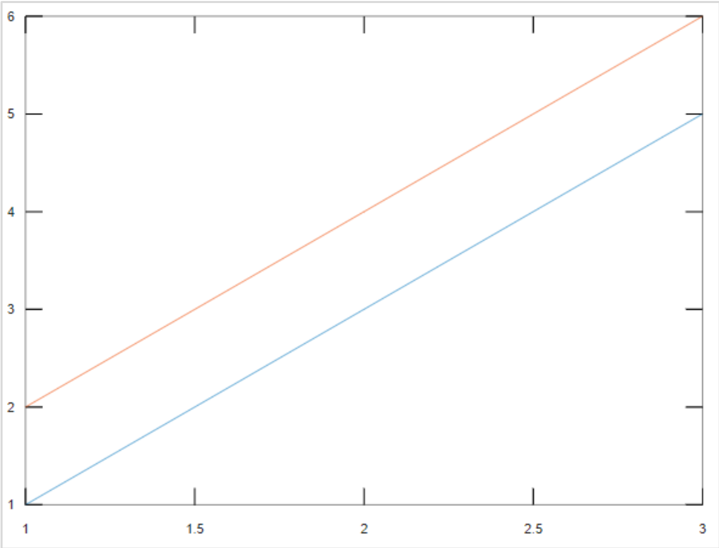
10. Rank with plotting:




10
8
6
4
2
0
1
1.5
2
2.5
3


10
8
6
4
2
0
1
1.5
2
2.5
3


6
5
4
3
2
1
1
1.2
1.4
1.6
1.8
2



Conclusion:

From this experiment we learn how to create One and Two-Dimensional Array (Row / Column Vector) and perform arithmetic operations like Addition, Subtraction, Multiplication and Exponentiation and matrix operations like Inverse, Transpose and Rank with plots.

Experiment: 2

Aim:

Performing Matrix Manipulations - Concatenating, Indexing, Sorting, Shifting, Reshaping, Resizing and Flipping about a Vertical Axis / Horizontal Axis; Creating Arrays X & Y of given size (1 x N) and Performing

(A). Relational Operations - >, <, ==, <=, >=, ~=

(B). Logical Operations - ~, &, |, XOR

Description:

1. Concatenating:

   Using square brackets to join existing matrices together and creating a matrix is called *concatenation.*

   I. Vertcat() is for vertical concatenation
   II. Horzcat() is for horizontal concatenation
   III. C = cat(dim,A,B) , concatenates B to the end of A along dimension dim when A and B have compatible sizes (the lengths of the dimensions match except for the operating dimension dim).
   IV. C = cat(dim,A1,A2,…,An) , concatenates A1, A2, … , An along dimension dim.

2. Indexing:

   In MATLAB, there are three primary approaches to accessing array elements based on their location (index) in the array. These approaches are indexing by position, linear indexing, and logical indexing.

3. Sorting:

   I. B = sort(A)
   II. B = sort(A) sorts the elements of A in ascending order.
   III. If A is a vector, then sort(A) sorts the vector elements.
   IV. If A is a matrix, then sort(A) treats the columns of A as vectors and sorts each column.
   V. If A is a multidimensional array, then sort(A) operates along the first array dimension whose size does not equal 1, treating the elements as vectors.

4. Circular Shifting:

   I. Y = circshift(A,K)
   II. Y = circshift(A,K) circularly shifts the elements in array A by K positions. If K is an integer, then circshift shifts along the first dimension of A whose size does not equal

5. Reshaping:

   I. B = reshape(A,sz)
   II. B = reshape(A,sz1,...,szN)

III. B = reshape(A,sz) reshapes A using the size vector, sz, to define size(B).

6. Resizing:
   Matrix can be resized just like indexing was done using colon and the rows and columns defined.

7. Flipping about a Vertical Axis / Horizontal Axis:
   I. B = fliplr(A)
      B = fliplr(A) returns A with its columns flipped in the left-right direction (that is, about a vertical axis).
   II. B = flipud(A)
      B = flipud(A) returns A with its rows flipped in the up-down direction (that is, about a horizontal axis).
8. Relational Operations:
   Relational operators compare the elements in two arrays and return logical true or false values to indicate where the relation holds.

==	Determine equality
>=	Determine greater than or equal to
>	Determine greater than
<=	Determine less than or equal to
<	Determine less than
~=	Determine inequality

9. Logical Operations:
   True or false (Boolean) conditions

<table>
<tr><td>&</td><td>Find logical AND</td></tr>
<tr><td>~</td><td>Find logical NOT</td></tr>
<tr><td>|</td><td>Find logical OR</td></tr>
<tr><td>xor</td><td>Find logical exclusive-OR</td></tr>
<tr><td>false</td><td>Logical 0 (false)</td></tr>
<tr><td>true</td><td>Logical 1 (true)</td></tr>
</table>

Code:

11. Matrix creation:

```
%clearing command prompt & variables resp
clc;clear all

%matrices
mat_1=[1 2 3]
mat_2=[4 1 6]
```

12. Concatenation:

```
%concatenation
a1=cat(1,mat_1,mat_2)%vertcat(mat_1,mat_2)%[mat_1 ; mat_2]
a2=cat(2,mat_1,mat_2)%horzcat(mat_1,mat_2)%[mat_1 mat_2]
```

13. Indexing:

```
%indexing
r1=a1(1,2)
r1a=a1(2,1)
r2=a1(1,:)
r3=a1(:,2)
```

14. Sorting:

```
%sorting
a3=sort(mat_2)  %array
a3a=sort(mat_2,'descend') %array
a4=sort(mat_3)  %matrix-column wise sorting
disp(mat_3)
a4a=sort(mat_3')' %row wise sort
```

15. Circular Shifting:

```
%circular shift
a5=circshift(mat_2,1)
a5a=circshift(mat_2',1)
a5b=circshift(mat_2,2)
a5c=circshift(mat_2',2)
```

16. Reshaping:

```
%reshape
a6=disp(a1)
a6a=reshape(a1,[3,2])
```

17. Resizing:

```
r4=a1(1:2,2:3) %resizing
```

18. Flipping about a Vertical Axis / Horizontal Axis:

```
%flipping
a7=disp(mat_3) %display original
a7a=fliplr(mat_3) %flip left right
a7b=flipud(mat_3) %flip up down
```

19. Relational Operations:

```
%relational operations
A=[1 2 3 4]
B=[-1 2 4 6]
rel1= A<3
rel2= A>3
rel3= A<=3
rel4= A<=3
rel5= A==3
rel6= A~=3
```

10. Logical Operations:

```
%logical operations
a8= 1 %true
a9= 0 %false
l1= and(a8,a9)
l1a= a8 & a9
l2= or(a8,a9)
l2a= a8 | a9
l3= not(a9)
l3a= ~a9
l4= xor(a8,a9)
```

```
mat_4= [0 0; 0 1; 1 0; 1 1]
x=mat_4(:,1)
y=mat_4(:,2)
And= and(mat_4(:,1),mat_4(:,2))
And1= x & y
Or= or(mat_4(:,1),mat_4(:,2))
Or1= x|y
N= ~mat_4(:,1)
N2=~mat_4(:,2)
N3=~x
N4=~y
Xor= xor(x,y)
Xor1=xor(mat_4(:,1),mat_4(:,2))
```

Results:

Total variables:

Vars
[1x4] A
[4x1]¬ And
[4x1]¬ And1
[1x4] B
[4x1]¬ N
[4x1]¬ N2
[4x1]¬ N3
[4x1]¬ N4
[4x1]¬ Or
[4x1]¬ Or1
[4x1]¬ Xor
[4x1]¬ Xor1
[2x3] a1
[1x6] a2
[1x3] a3
[1x3] a3a
[2x3] a4
[2x3] a4a
[1x3] a5
[3x1] a5a
[1x3] a5b

Vars
[1x3] a5b
[3x1] a5c
(abc) a6
[3x2] a6a
(abc) a7
[2x3] a7a
[2x3] a7b
a8
a9
¬ l1
¬ l1a
¬ l2
¬ l2a
¬ l3
¬ l3a
¬ l4
[1x3] mat_1
[1x3] mat_2
[2x3] mat_3
[4x2] mat_4
r1

r1
r1a
[1x3] r2
[2x1] r3
[2x2] r4
[1x4]¬ rel1
[1x4]¬ rel2
[1x4]¬ rel3
[1x4]¬ rel4
[1x4]¬ rel5
[1x4]¬ rel6
[4x1] x
[4x1] y

Matrix creation:

```
mat_1 =

   1   2   3

mat_2 =

   4   1   6
```

Concatenation:

```
a1 =

   1   2   3
   4   1   6

a2 =

   1   2   3   4   1   6
```

Indexing:

```
r1 = 2
r1a = 4
r2 =

   1   2   3

r3 =

   2
   1
```

Sorting:

```
a3 =

   1   4   6

a3a =

   6   4   1

a4 =

   1   4   2
   4   6   2

   4   6   2
   1   4   2
a4a =

   2   4   6
   1   2   4
```

Circular Shifting:

```
a5 =

   6   4   1

a5a =

   6
   4
   1

a5b =

   1   6   4

a5c =

   1
   6
   4
```

Reshaping:

```
a6 =     1     2     3
   4     1     6

a6a =

   1     1
   4     3
   2     6
```

Resizing:

```
r4 =

    2     3
    1     6
```

Flipping about a Vertical Axis / Horizontal Axis:

```
a7 =     4     6     2
   1     4     2

a7a =

   2     6     4
   2     4     1
```

Relational Operations:

```
A =

   1   2   3   4

B =

  -1   2   4   6

rel1 =

  1  1  0  0

rel2 =

  0  0  0  1
```

```
rel3 =

  1  1  1  0

rel4 =

  1  1  1  0

rel5 =

  0  0  1  0

rel6 =

  1  1  0  1
```

Logical Operations:

```
a8 = 1
a9 = 0
l1 = 0
l1a = 0
l2 = 1
l2a = 1
l3 = 1
l3a = 1
l4 = 1
```

```
mat_4 =

   0   0
   0   1
   1   0
   1   1

x =

   0
   0
   1
   1

y =

   0
   1
   0
   1
```

```
And =

  0
  0
  0
  1

And1 =

  0
  0
  0
  1

Or =

  0
  1
  1
  1

Or1 =

  0
  1
  1
  1
```

```
N =

  1
  1
  0
  0

N2 =

  1
  0
  1
  0

N3 =

  1
  1
  0
  0
```

```
N4 =

  1
  0
  1
  0

Xor =

  0
  1
  1
  0

Xor1 =

  0
  1
  1
  0
```

Conclusion:

From this experiment we learn how to perform Matrix Manipulations like Concatenating, Indexing, Sorting, Shifting, Reshaping, Resizing and Flipping about a Vertical Axis / Horizontal Axis and Creating Arrays X & Y of given size (1 x N) and performing Relational Operations like >, <, ==, <=, >=, ~= and logical Operations like ~, &, |, XOR

Experiment: 3

Aim:

Generating a set of Commands on a given Vector (Example: X = [1 8 3 9 0 1]) to

(A). Add up the values of the elements (Check with sum)

(B). Compute the Running Sum (Check with sum), where Running Sum for element j = the sum of the elements from 1 to j, inclusive.

(C) Generating a Random Sequence using rand() / randn() functions and plot them.

Description:

1. Sum:

   S = sum(A)

   I. S = sum(A) returns the sum of the elements of A along the first array dimension whose size does not equal 1.
   II. If A is a vector, then sum(A) returns the sum of the elements.
   III. If A is a matrix, then sum(A) returns a row vector containing the sum of each column.

2. Cumulative sum:

   B = cumsum(A)

   I. B = cumsum(A) returns the cumulative sum of A starting at the beginning of the first array dimension in A whose size does not equal 1.
   II. If A is a vector, then cumsum(A) returns a vector containing the cumulative sum of the elements of A.
   III. If A is a matrix, then cumsum(A) returns a matrix containing the cumulative sums for each column of A.

3. Multiple plots in same figure:
   Create a line plot of both sets of data using plot function.

4. Random numbers:

   For uniform distribution: X = rand
   For normal distribution: X = rand(n)

   I. X = rand returns a single uniformly distributed random number in the interval (0,1).
   II. X = rand(n) returns an n-by-n matrix of random numbers.
   III. X = rand(sz1,...,szN) returns an sz1-by-...-by-szN array of random numbers where sz1,...,szN indicate the size of each dimension. For example, rand(3,4) returns a 3-by-4 matrix.
   IV. X = rand(sz) returns an array of random numbers where size vector sz specifies size(X). For example, rand([3 4]) returns a 3-by-4 matrix.

5. Historgram:

hist(x)

I. hist(x) creates a histogram bar chart of the elements in vector x. The elements in x are sorted into 10 equally spaced bins along the *x*-axis between the minimum and maximum values of x. hist displays bins as rectangles, such that the height of each rectangle indicates the number of elements in the bin.

II. If the input is a multi-column array, hist creates histograms for each column of x and overlays them onto a single plot.

III. If the input is of data type categorical, each bin is a category of x.

Code:

1. Sum and cumulative sum:

```
% Sum and Cumulative sum of a vector
x=[1 8 3 9 0 1]
s1=sum(x)
s2=cumsum(x)
figure;plot(x)
ylabel('x')  %labelling y axis
figure;plot(s1)
ylabel('sum')
figure;plot(s2)
ylabel('cum. sum')
```

2. Multiple plots in same figure:

```
%multiple plots in same figure window
y=[x',s2'];
figure;plot(y)
ylabel('multiple plots') %label at y
title('Combine Plots') %title
legend('x','cumsum(x)') %box to identify colour line

%alternate way of plotting multiple
figure;plot(1:6,x,1:6,s2)
title('Combine Plots alternate method')
```

3. Random numbers:

Uniform distribution :

```
%random numbers
%uniform distribution
a=rand
b=rand(3)
c=rand(1,5)
```

Normal distribution :

```
%normal distribution
a1=randn
b1=randn(3)
c1=randn(1,5)
```

4. Plot and histogram for uniform distribution :

```
%generate long sequence
d=rand(1,1000);
%uniform distribution plot
figure;plot(d)
title('long sequence')
%histogram
figure;hist(d)
title('Histogram')
```

5. Plot and histogram for normal distribution :

```
d1=randn(1,1000);
%normal distribution plot
figure;plot(d1)
title('long sequence 2')
%histogram
figure;hist(d1)
title('Histogram 2')
```

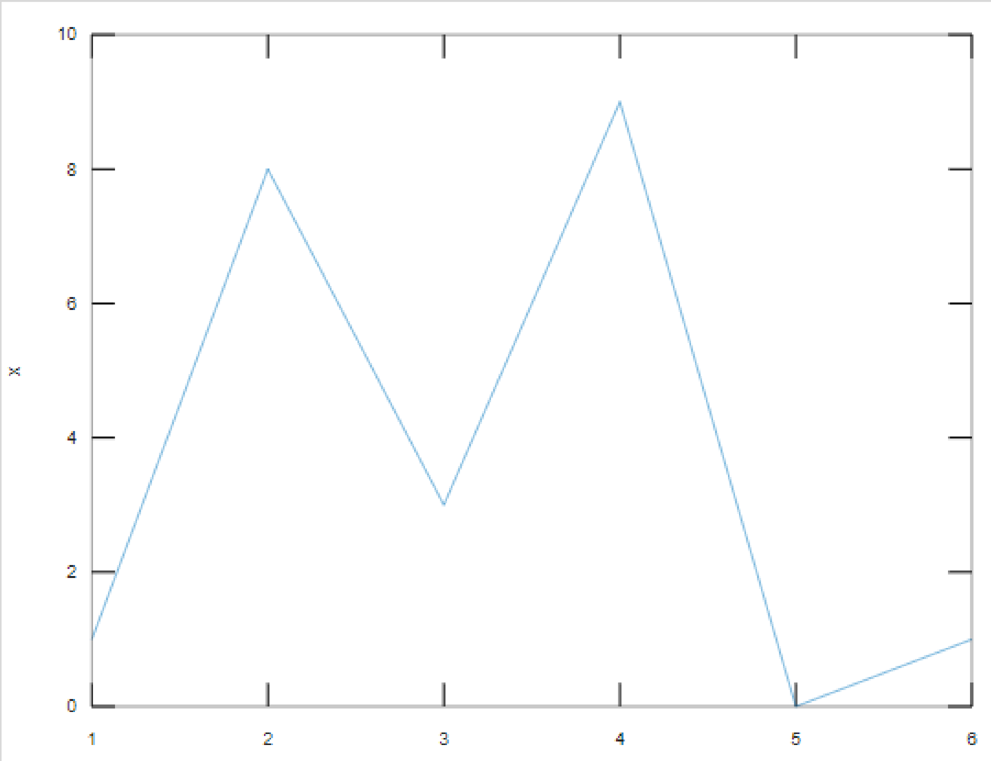
Results:

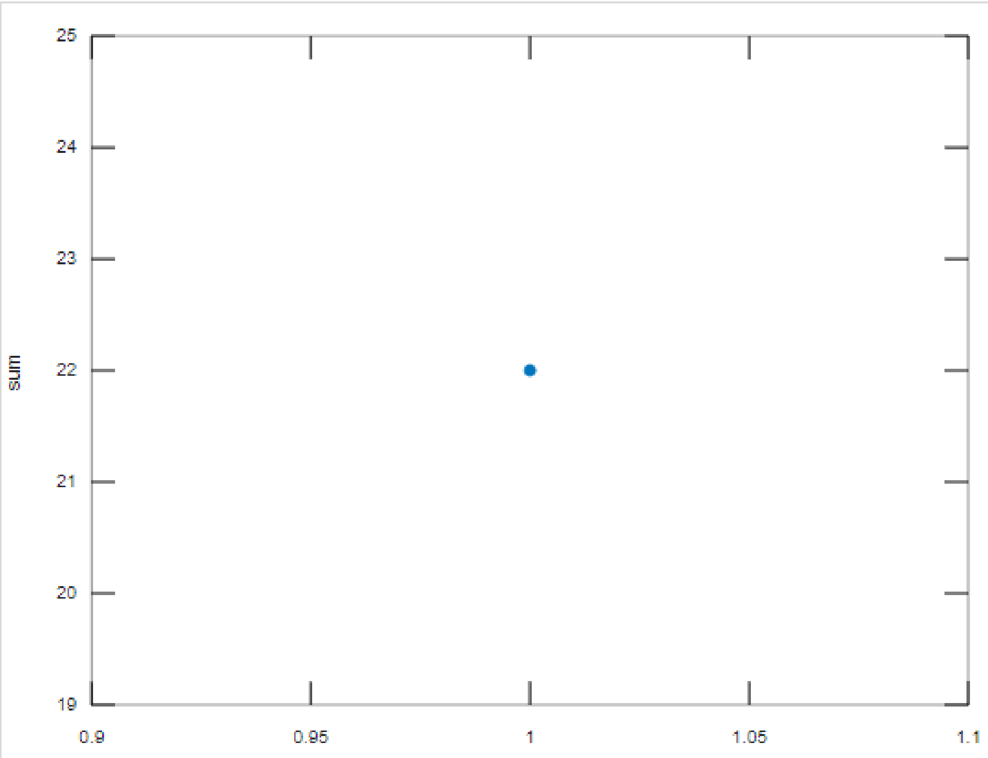
Total variables:

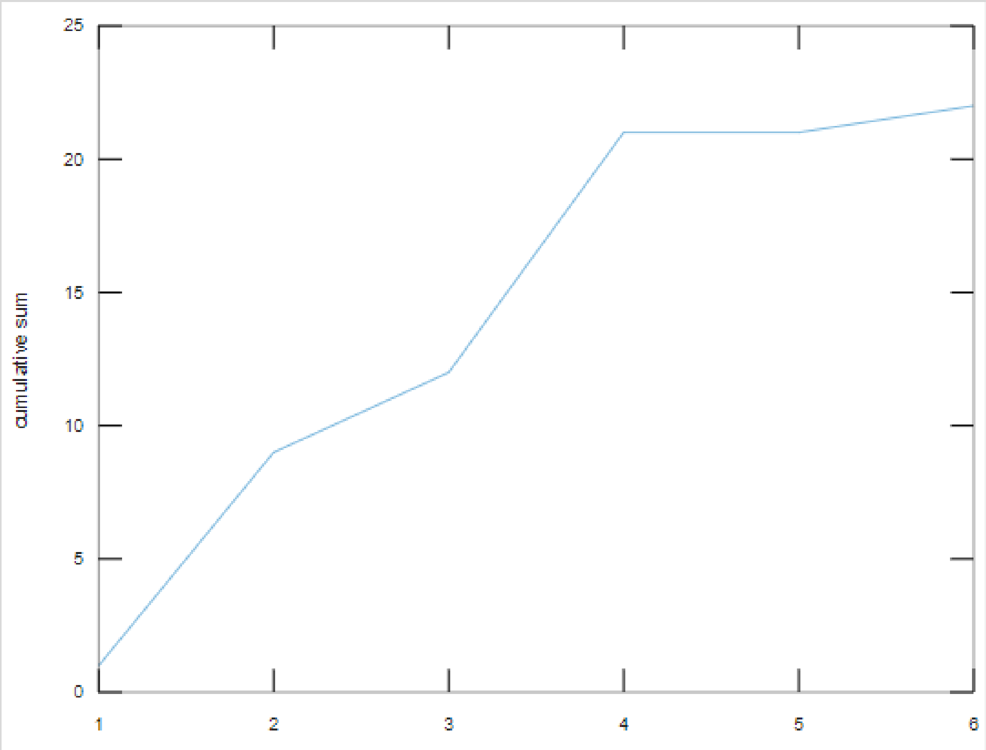
Vars
a
a1
ans
[3x3] b
[3x3] b1
[1x5] c
[1x5] c1
[1x1000] d
[1x1000] d1
s1
[1x6] s2
[1x6] x
[6x2] y

Sum and cumulative sum:

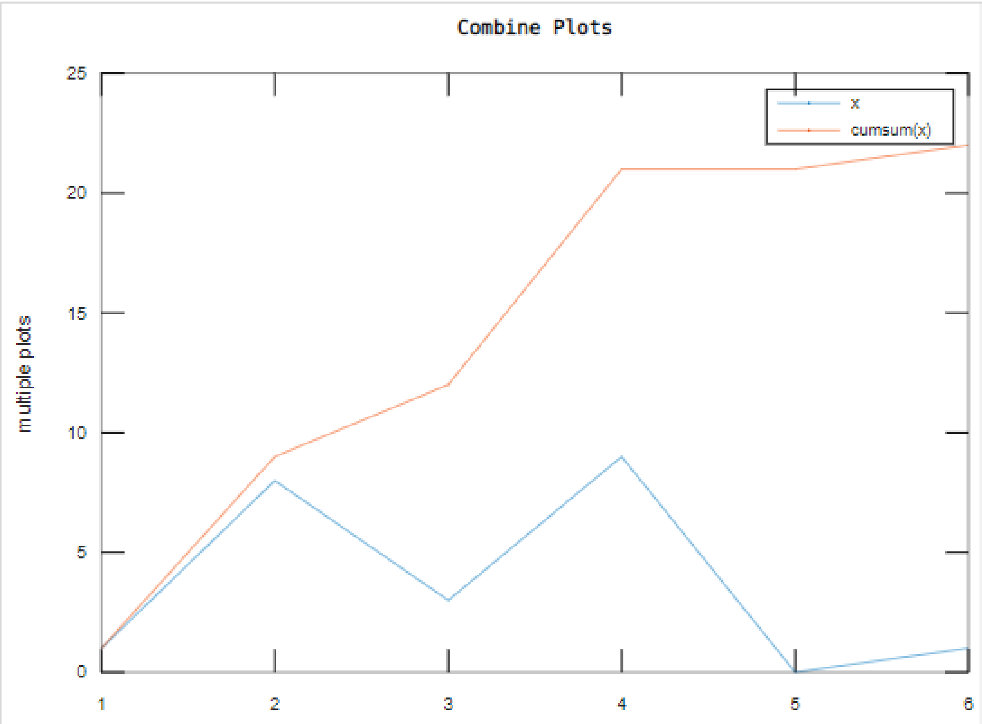
```
x =

   1   8   3   9   0   1

s1 = 22
s2 =

    1    9   12   21   21   22
```

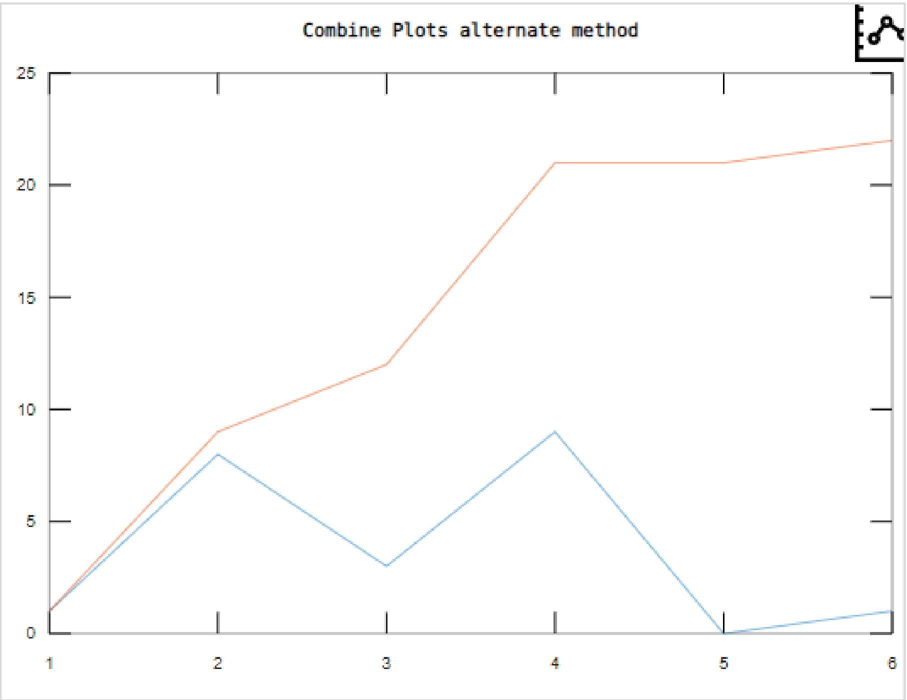






Multiple plots in same figure:





Random numbers:

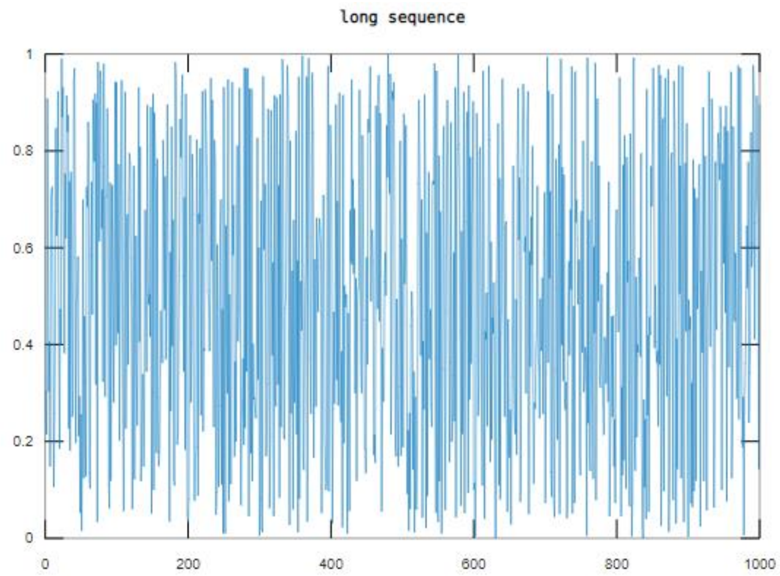
Uniform distribution :

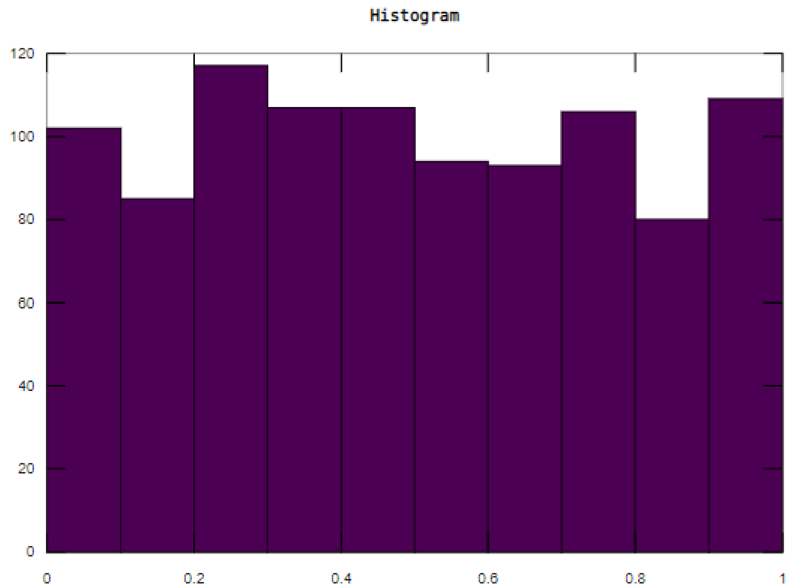
```
a = 0.7675
b =

   0.9820   0.3664   0.2433
   0.5538   0.1554   0.5246
   0.6607   0.9455   0.2960

c =

   0.5174   0.7633   0.8747   0.5325   0.5467
```

Normal distribution:

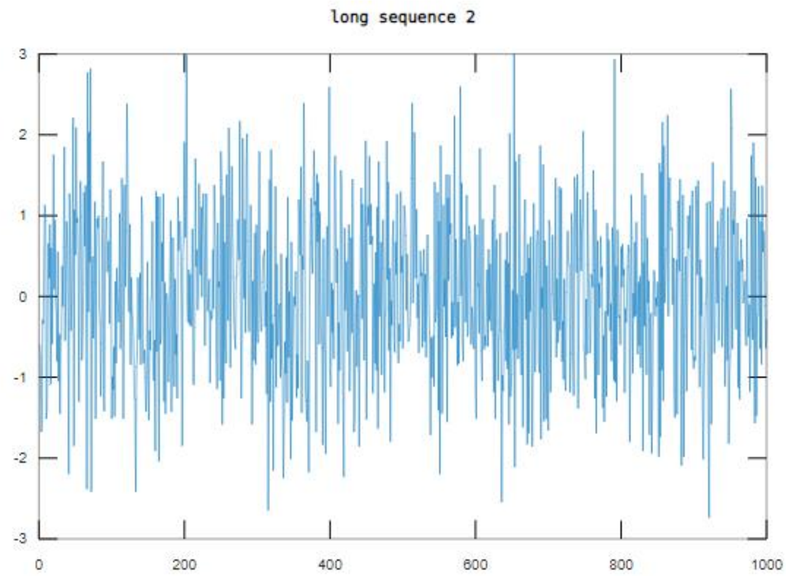
```
a1 = 0.5456
b1 =

  -0.1865  -0.6204  -0.4517
  -0.4491  -2.8066   0.3738
  -0.9071  -0.7826  -0.3081

c1 =

  -0.7833  -0.7409   1.3177  -1.7298   1.3696
```

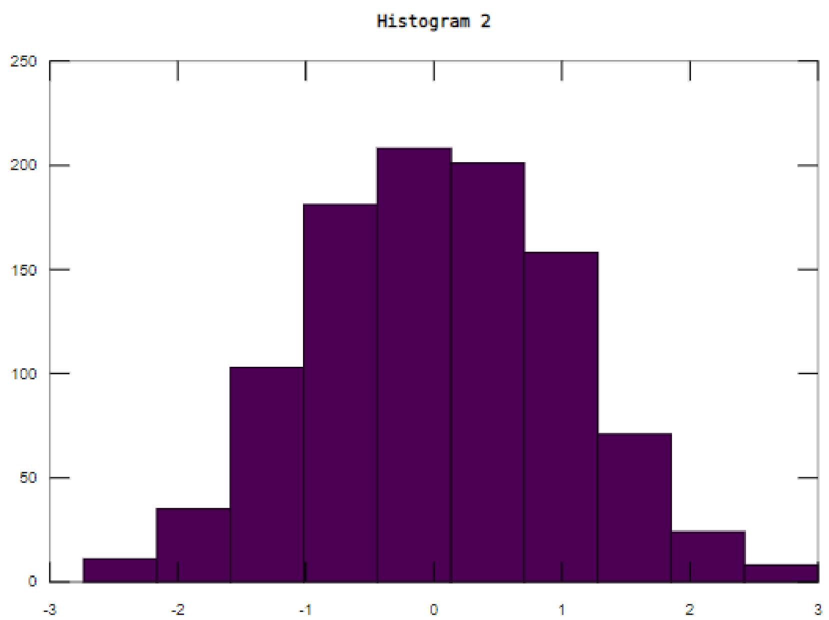
Plot and histogram for uniform distribution :





Plot and histogram for normal distribution:





Conclusion:

From this experiment we learn how to generate a set of Commands on a given Vector (Example: X = [1 8 3 9 0 1]) to Add up the values of the elements (Check with sum), Compute the Running Sum (Check with sum), where Running Sum for element j = the sum of the elements from 1 to j, inclusive and Generating a Random Sequence using rand() / randn() functions and plot them.

Experiment: 4

Aim:

Generating a set of Commands on a given Vector to-

A. Evaluating a given expression and rounding it to the nearest integer value using Round, Floor, Ceil and fix functions.
B. Trigonometric functions= sin(t), cos(t), tan(t), sec(t), cosec(t), cot(t) for a given duration, 't'.
C. Logarithmic and other functions= log(A), log10(A), square root of A and real nth root of A.

Description:

1) Round

   Y = round(X)

   I. Y = round(X) rounds each element of X to the nearest integer. In the case of a tie, where an element has a fractional part of exactly 0.5, the round function rounds away from zero to the integer with larger magnitude.

   II. Y = round(X,N) rounds to N digits:

      a) N > 0: round to N digits to the *right* of the decimal point.
      b) N = 0: round to the nearest integer.
      c) N < 0: round to N digits to the *left* of the decimal point.

2) Floor

   Y = floor(X)

   Y = floor(X) rounds each element of X to the nearest integer less than or equal to that element.

3) Ceil

   Y = ceil(X)

   Y = ceil(X) rounds each element of X to the nearest integer greater than or equal to that element.

4) Fix

   Y = fix(X)

   Y = fix(X) rounds each element of X to the nearest integer toward zero. For positive X, the behavior of fix is the same as floor. For negative X, the behavior of fix is the same as ceil.

5) Logarithmic functions

   I. Y = log(X)

      Y = log(X) returns the natural logarithm ln(x) of each element in array X.

   II. Y = log2(X)

      Y = log2(X) computes the base 2 logarithm of the elements of X such that $2^Y = X$.

   III. Y = log10(X)

Y = log10(X) returns the common logarithm of each element in array X. The function accepts both real and complex inputs. For real values of X in the interval (0, Inf), log10 returns real values in the interval (-Inf ,Inf). For complex and negative real values of X, the log10 function returns complex values.

IV. Y = exp(X)

Y = exp(X) returns the exponential e^x for each element in array X.

6) Other functions- square root of A and real nth root of A.

I. B = sqrt(X)

B = sqrt(X) returns the square root of each element of the array X. For the elements of X that are negative or complex, sqrt(X) produces complex results.

II. Y = nthroot(X,N)

Y = nthroot(X,N) returns the real nth root of the elements of X.

Both X and N must be real scalars or arrays of the same size. If an element in X is negative, then the corresponding element in N must be an odd integer.

7) Trigonometric functions

I. Y = sin(X)

Y = sin(X) returns the sine of the elements of X. The sin function operates elementwise on arrays. The function accepts both real in the interval [-1, 1] and complex inputs.

II. Y = cos(X)

Y = cos(X) returns the cosine for each element of X. The cos function operates elementwise on arrays. The function accepts both real in the interval [-1, 1] and complex inputs.

III. Y = tan(X)

Y = tan(X) returns the tangent of each element of X. The tan function operates elementwise on arrays. The function accepts both real in the interval [-∞, ∞] and complex inputs.

IV. Y = cot(X)

Y = cot(X) returns the cotangent of elements of X. The cot function operates elementwise on arrays. The function accepts both real in the interval [-∞, ∞] and complex inputs.

V. Y = sec(X)

Y = sec(X) returns the secant of the elements of X. The sec function operates elementwise on arrays. The function accepts both real in the interval [-∞, -1] and [1, ∞] and complex inputs.

VI. Y = csc(X)

Y = csc(X) returns the cosecant of the elements of X. The csc function operates elementwise on arrays. The function accepts both real in the interval [-∞, -1] and [1, ∞] and complex inputs.

Code:

1. Round, Floor, Ceil and fix:

```
%rounding functions
a=[-1.6 1.1 1.6]
a1=round(a)
a2=floor(a)
a3=ceil(a)
a4=fix(a)
```

2. Logarithmic functions

```
%logarithmic functions and other functions
b=[e 10 2 5]
b1=log(b)
b2=log10(b)
b3=log2(b)
%OR
c=[exp(1) 10 2 5]
c1=log(c)
c2=log10(c)
c3=log2(c)
%other log functions using logaB property
c4=log(c)/log(5)
c5=log10(c)/log10(5)
c6=log2(c)/log2(5)
```

3. Other functions- square root of A and real nth root of A.

```
%square root
d=sqrt(100)
n=2;
d1=nthroot(100,n) %nth root
```

4. Trigonometric functions
   a) Sin(t)

```
%Trignometric functions and plotting
%sin
x= 0:pi/100:5*pi; %x axis
y=sin(x); %y axis
figure;plot(x./pi,y) %plotting
xlabel('theta*pi(rad)')
title('sin function')
```

   b) Cos(t)

```
%cos
x= 0:pi/100:5*pi; %x axis
y=cos(x); %y axis
figure;plot(x./pi,y) %plotting
xlabel('theta*pi(rad)')
title('cos function')
```

   c) Tan(t)

```
%tan
x= 0:pi/100:5*pi; %x axis
y=tan(x); %y axis
figure;plot(x./pi,y) %plotting
ylim([-5 5])
xlabel('theta*pi(rad)')
title('tan function')
```

d) Cot(t)

```
%cot
x= 0:pi/100:5*pi; %x axis
y=cot(x); %y axis
figure;plot(x./pi,y) %plotting
ylim([-5 5])
xlabel('theta*pi(rad)')
title('cot function')
```

e) Secant(t)

```
%secant
x= 0:pi/100:5*pi; %x axis
y=sec(x); %y axis
figure;plot(x./pi,y) %plotting
ylim([-5 5])
xlabel('theta*pi(rad)')
title('sec function')
```

f) Cosecant(t)

```
%cosecant
x= 0:pi/100:5*pi; %x axis
y=csc(x); %y axis
figure;plot(x./pi,y) %plotting
ylim([-5 5])
xlabel('theta*pi(rad)')
title('cosecant function')
```

Results:

Total variables:

```
Vars
[1x3] a
[1x3] a1
[1x3] a2
[1x3] a3
[1x3] a4
[1x4] b
[1x4] b1
[1x4] b2
[1x4] b3
[1x4] c
[1x4] c1
[1x4] c2
[1x4] c3
[1x4] c4
[1x4] c5
[1x4] c6
# d
# d1
# n
[1x501] x
[1x501] y
```

Round, Floor, Ceil and fix:

```
a =

  -1.6000    1.1000    1.6000

a1 =

  -2   1   2

a2 =

  -2   1   1
```

```
a3 =

  -1   2   2

a4 =

  -1   1   1
```

Logarithmic functions:

```
b =

    2.7183   10.0000    2.0000    5.0000

b1 =

   1.0000   2.3026   0.6931   1.6094

b2 =

   0.4343   1.0000   0.3010   0.6990

b3 =

   1.4427   3.3219   1.0000   2.3219

c =

    2.7183   10.0000    2.0000    5.0000

c1 =

   1.0000   2.3026   0.6931   1.6094
```

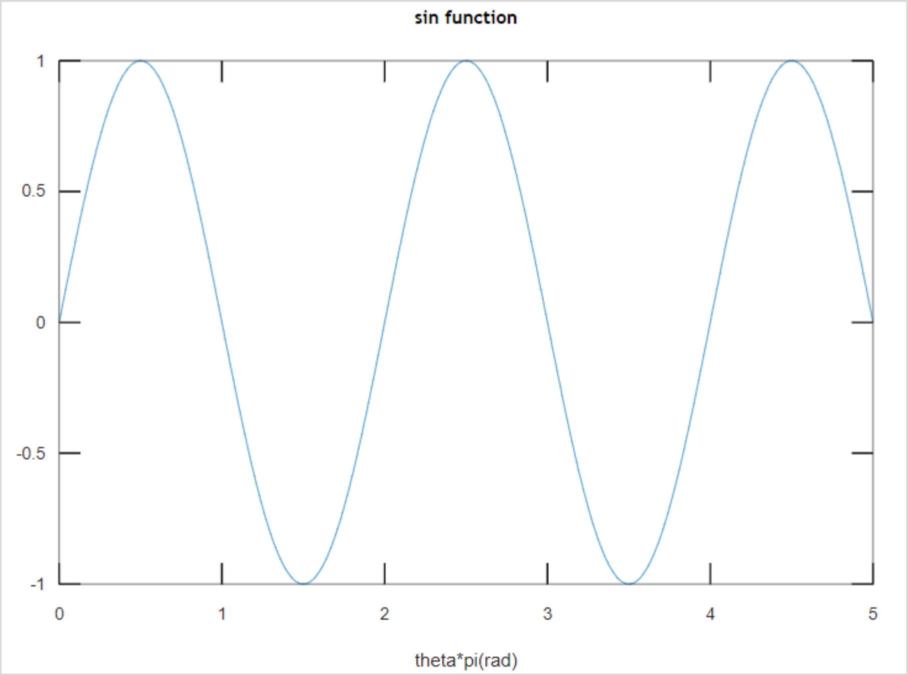
```
c2 =

   0.4343   1.0000   0.3010   0.6990

c3 =

   1.4427   3.3219   1.0000   2.3219

c4 =

   0.6213   1.4307   0.4307   1.0000

c5 =

   0.6213   1.4307   0.4307   1.0000

c6 =

   0.6213   1.4307   0.4307   1.0000
```

Other functions- square root of A and real nth root of A :

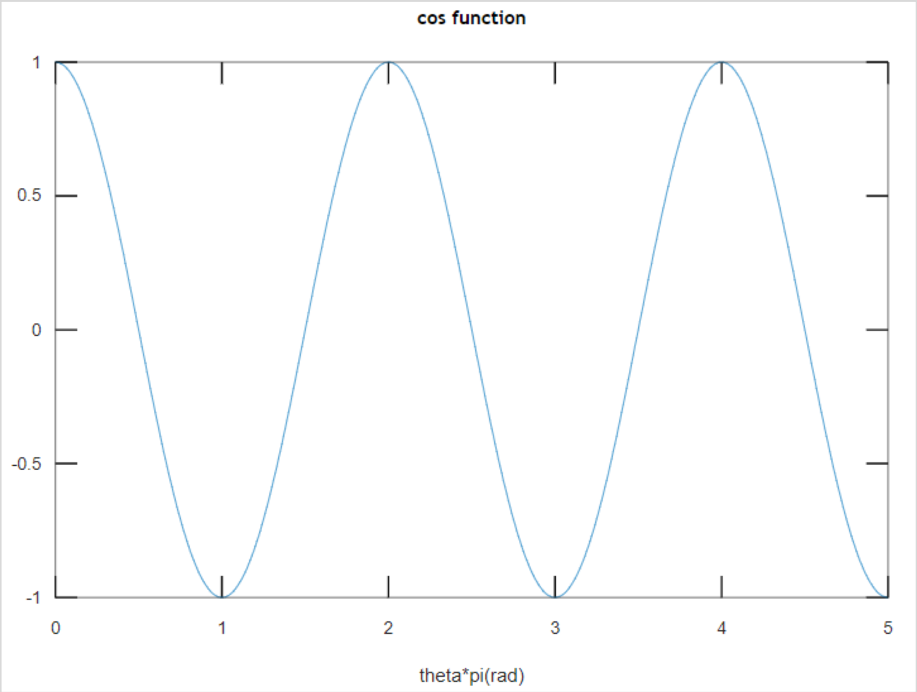
```
d = 10
d1 = 10
```

Trigonometric functions:

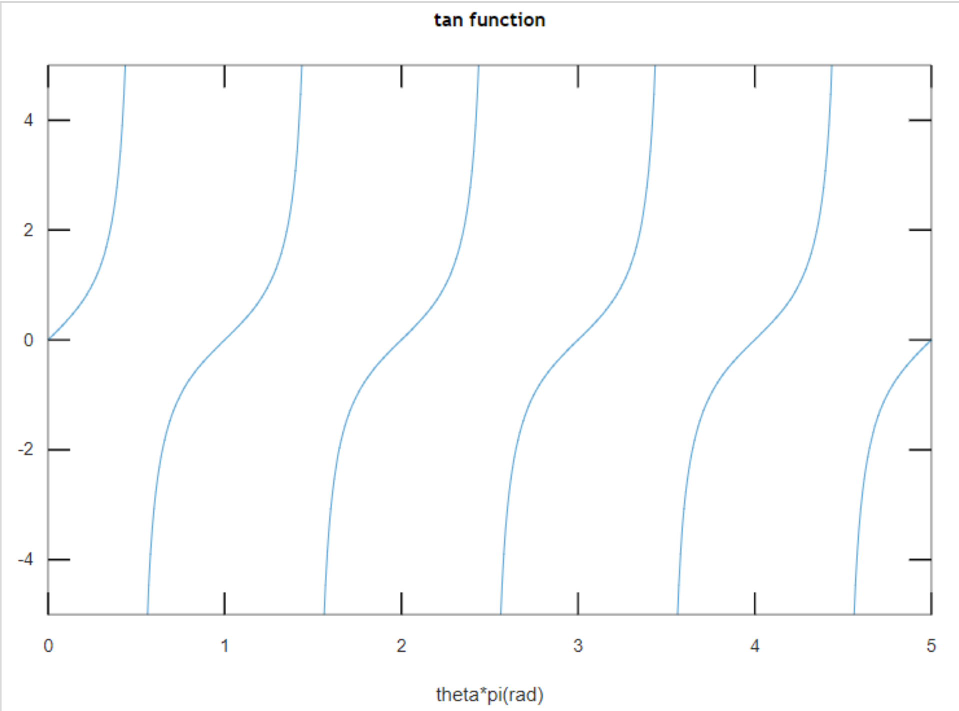
a. Sin(t)



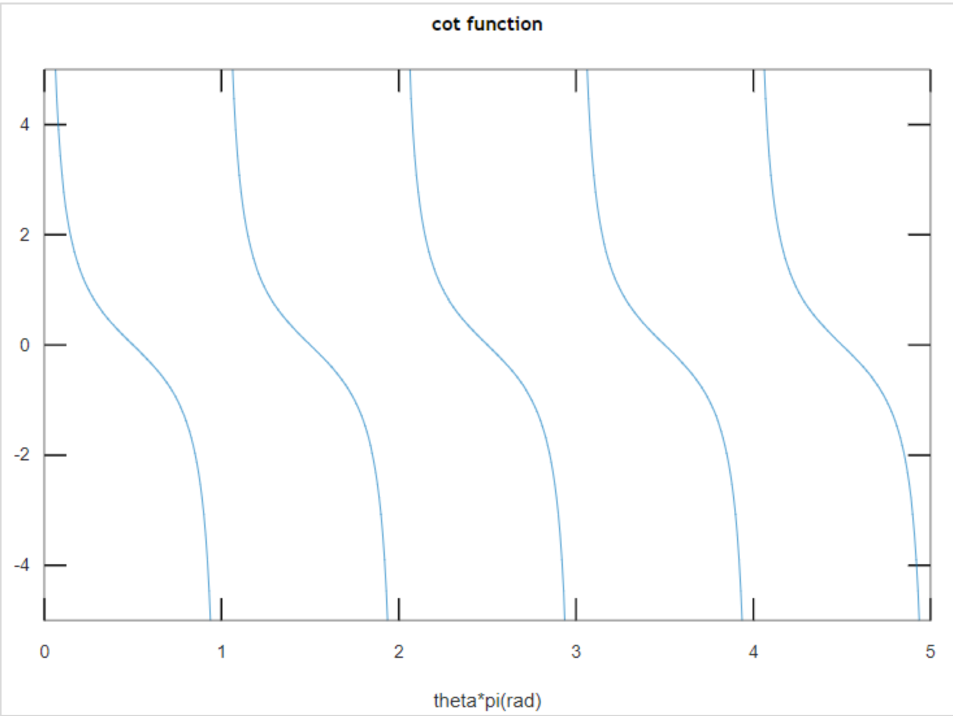
b. Cos(t)



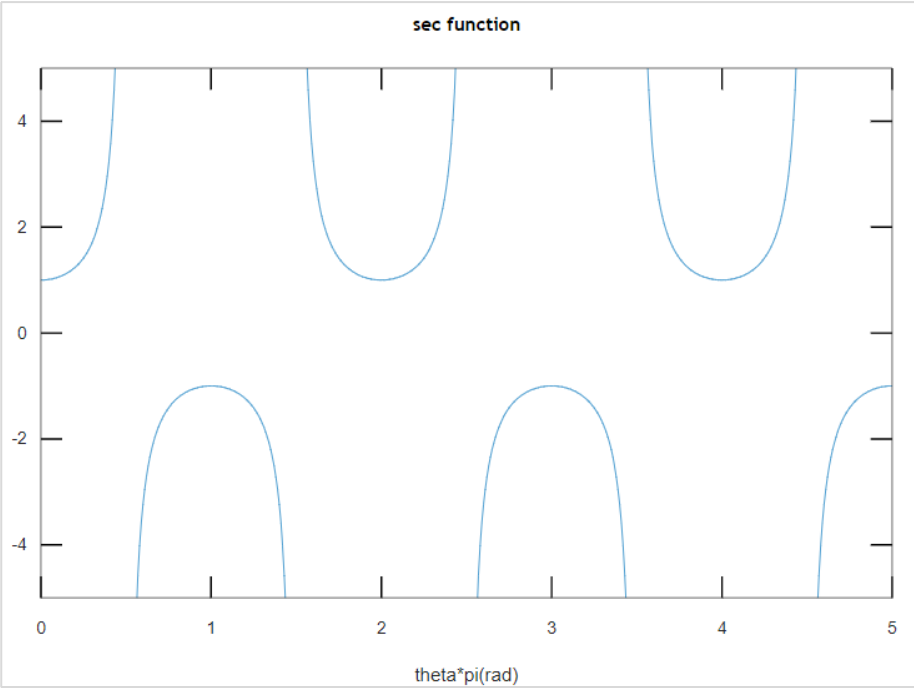
c. Tan(t)



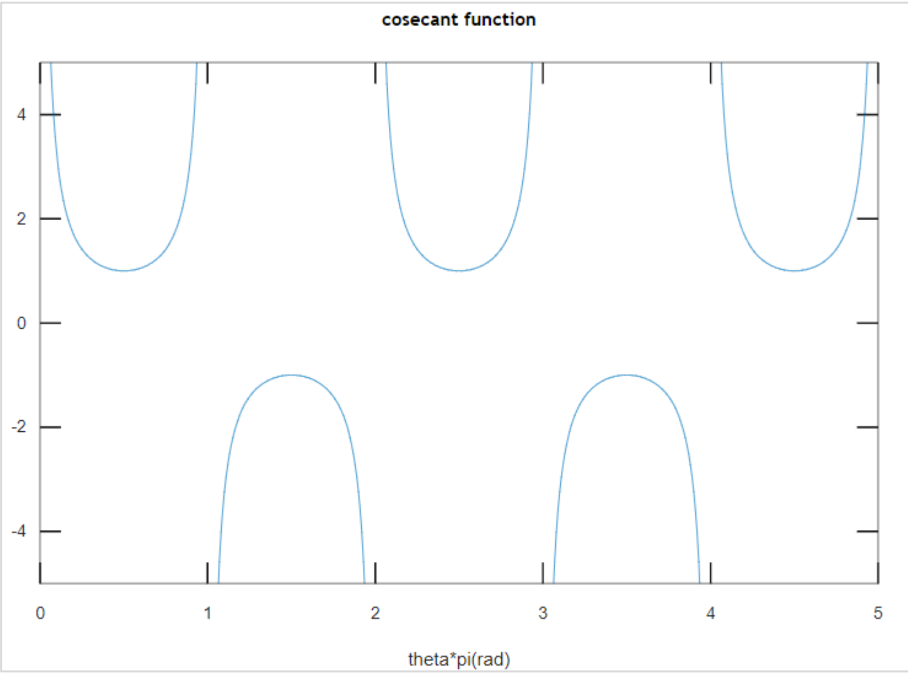
d. Cot(t)



e. Secant(t)



f. Cosecant(t)



Conclusion:

From this experiment we learn how to generate a set of Commands on a given Vector to evaluate a given expression and rounding it to the nearest integer value using Round, Floor, Ceil and fix functions, trigonometric functions like sin(t), cos(t), tan(t), sec(t), cosec(t), cot(t) for a given duration, ’t’ and logarithmic and other functions= log(A), log10(A), square root of A and real nth root of A.

Experiment: 5

Aim:

Generating a set of Commands on a given Vector to-

c. Create a vector Z with elements, $Z = \frac{(-1)^{n+1}}{2n-1}$

   Add up to 100 elements of the vector Z. Plot Z

d. Plot the functions, $x, x^3, e^x, \exp(x^2)$ over the interval $0<x<4$ (by choosing appropriate mesh values for x to obtain smooth curves), on a rectangular plot.

Description:

1. Y = exp(X)

   Y = exp(X) returns the exponential e^x for each element in array X.

2. Sum:

   S = sum(A)

   I. S = sum(A) returns the sum of the elements of A along the first array dimension whose size does not equal 1.
   II. If A is a vector, then sum(A) returns the sum of the elements.
   III. If A is a matrix, then sum(A) returns a row vector containing the sum of each column.

3. Multiple plots in same figure:
   Create a line plot of both sets of data using plot function.

4. Arithmetic operations:
   I. Addition: +
   II. Subtraction: -
   III. Multiplication: *
   IV. Element-wise multiplication: **.***
   V. Exponentiation:
      Y = exp(X) ,which returns the exponential e^x for each element in array X.

5. Subplot
   subplot(m,n,p)
   subplot(m,n,p) divides the current figure into an m-by-n grid and creates axes in the position specified by p. MATLAB® numbers subplot positions by row. The first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on. If axes exist in the specified position, then this command makes the axes the current axes.

Code:

1) Vector Z with elements, $Z = \frac{(-1)^{n+1}}{2n-1}$ and plotting Z

```
%create vectors,average and plot
n=[1:100];
z=(-1).^(n+1)./(2*n-1);
xlabel('n')
ylabel('z')
figure;plot(n,z,'linewidth',2)
```

2) Add 100 elements of the vector Z

```
%sum of all values in z
s=sum(z)
```

3) Plot functions, $x, x^3, e^x, \exp(x^2)$ over the interval $0<x<4$

```
%plot functiions
x=[0:0.1:4];
figure;plot(x,'linewidth',2)
figure;plot(x,x.^3,'linewidth',2)
figure;plot(x,exp(x),'linewidth',2)
figure;plot(x,exp(x.^2),'linewidth',2)
```

4) Using subplots and plotting the functions

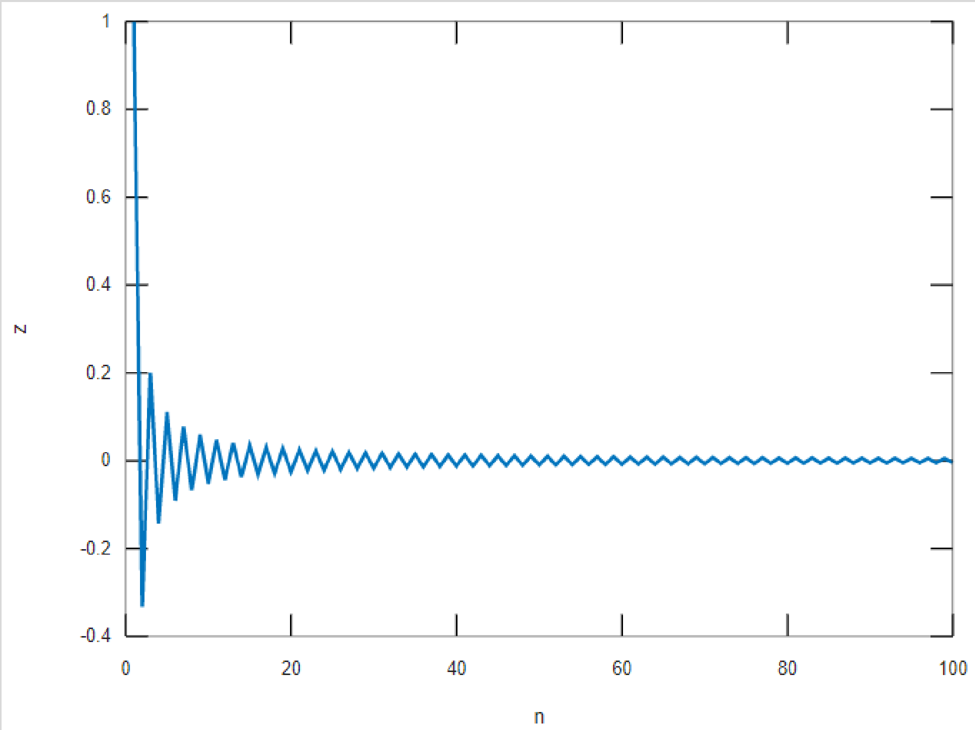
```
%using subplots
subplot(2,2,1);
plot(x,'linewidth',2)
subplot(2,2,2);
plot(x,x.^3,'linewidth',2)
subplot(2,2,3);
plot(x,exp(x),'linewidth',2)
subplot(2,2,4);
plot(x,exp(x.^2),'linewidth',2)
```

Results:

Total variables:

Vars
[1x100] n
s
[1x41] x
[1x100] z

Vector Z with elements, $Z = (-1)^{n+1}/2n-1$ and plotting Z:

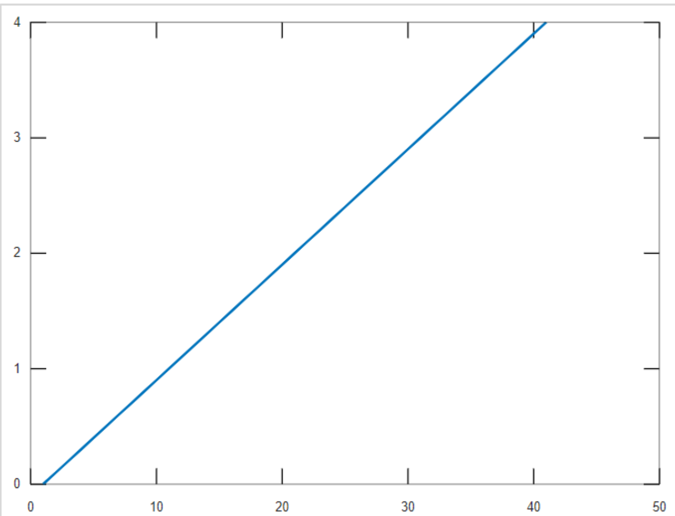


Add 100 elements of the vector Z:

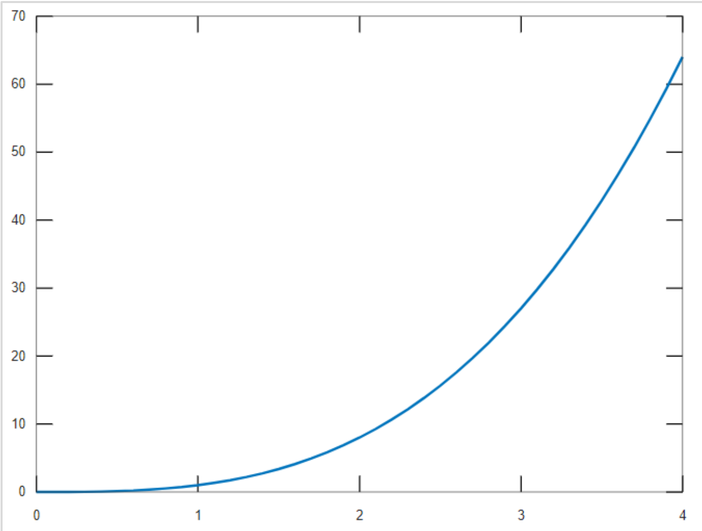
```
s = 0.7829
```

Plot functions, $x, x^3, e^x, \exp(x^2)$ over the interval $0<x<4$:

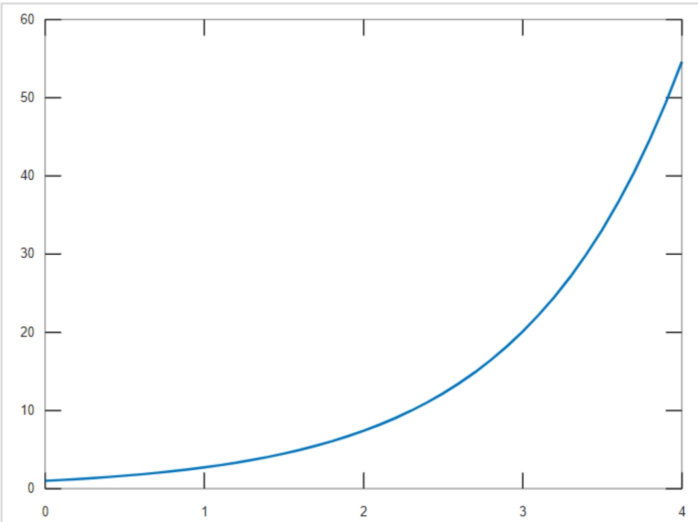
I. x



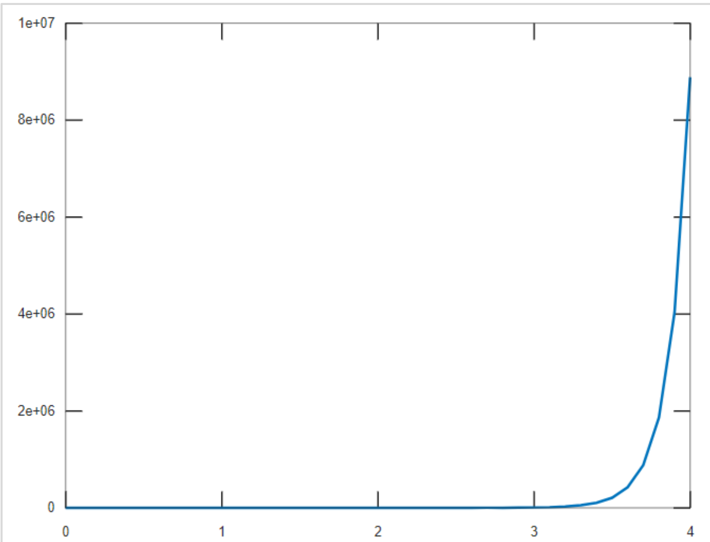
II. x^3



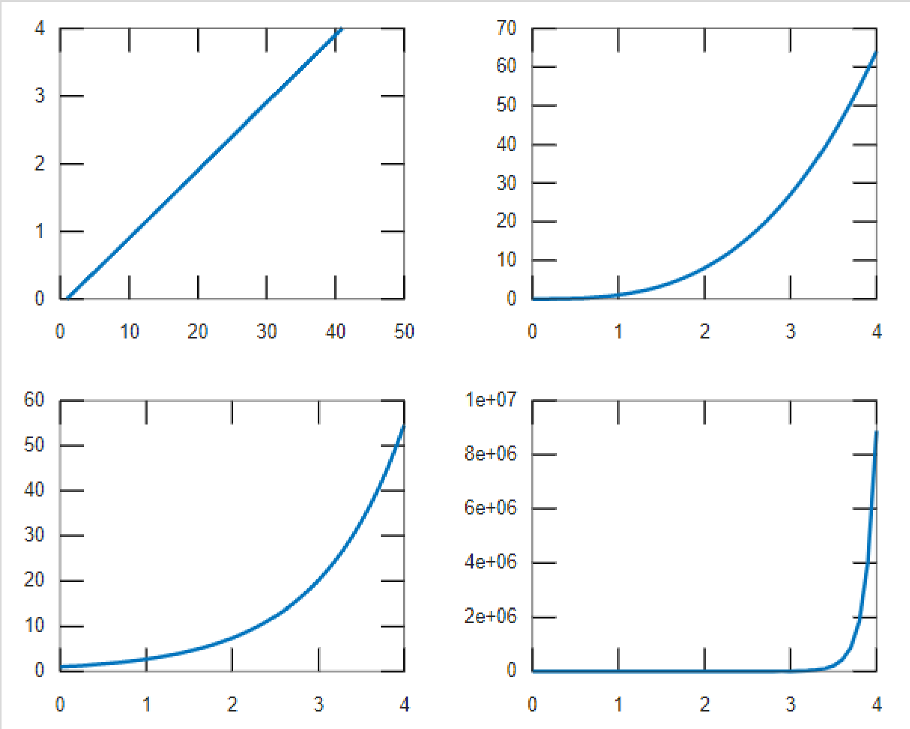
III. e^x



IV. $\exp(x^2)$



Using subplots and plotting the functions :



Conclusion:

From this experiment we learn how to generate a set of Commands on a given Vector to create a vector Z with elements, Z= $(-1)^{n+1}/2n-1$ and add up to 100 elements of the vector Z. Plot Z and plot the functions, $x, x^3, e^x, \exp(x^2)$ over the interval $0<x<4$ (by choosing appropriate mesh values for x to obtain smooth curves), on a rectangular plot.

Experiment: 6

Aim:

Generating a set of Commands on a given Vector to-

a. Generating a Sinusoidal Signal of a given frequency with Titling, Labelling, Adding Text, Adding Legends, Printing Text in Greek Letters, Plotting as Multiple and Subplot.
b. Time scale the generated signal for different values. E.g. 2X, 4X, 0.25X, 0.0625X.

Description:

1) Title
   title(txt)
   title(txt) adds the specified title to the axes or chart returned by the gca command. Reissuing the title command causes the new title to replace the old title.
2) Label

xlabel()	Label x-axis
ylabel()	Label y-axis
zlabel()	Label z-axis

3) Adding Text
   text(x,y,txt)
   text(x,y,z,txt)
   I. text(x,y,txt) adds a text description to one or more data points in the current axes using the text specified by txt. To add text to one point, specify x and y as scalars. To add text to multiple points, specify x and y as vectors with equal length.
   II. text(x,y,z,txt) positions the text in 3-D coordinates.
4) Legends
   legend(label1,...,labelN)
   I. legend creates a legend with descriptive labels for each plotted data series.The legend automatically updates when you add or delete data series from the axes. This command creates a legend for the current axes or chart returned by gca. If the current axes are empty, then the legend is empty. If axes do not exist, then this command creates them.
   II. legend(label1,...,labelN) sets the legend labels. Specify the labels as a list of character vectors or strings, such as legend('Jan','Feb','Mar').
5) Greek letter:
   Create a simple line plot and add a title. Include the Greek letter π in the title by using the TeX markup \pi.

^{ }	Superscript
_{ }	Subscript
\bf	Bold font
\it	Italic font
\sl	Oblique font (usually the same as italic font)
\rm	Normal font
\fontname{*specifier*}	Font name — Replace *specifier* with the name of a font family. You can use this in combination with other modifiers.
\fontsize{*specifier*}	Font size —Replace *specifier* with a numeric scalar value in point units.
\color{*specifier*}	Font color — Replace *specifier* with one of these colors: red, green, yellow, magenta, blue, black, white, gray, darkGreen, orange, or lightBlue.
\color[rgb]{specifier}	Custom font color — Replace *specifier* with a three-element RGB triplet.

6) Multiple plots
Create a line plot of both sets of data using plot function.
7) Subplots
subplot(m,n,p)
subplot(m,n,p) divides the current figure into an m-by-n grid and creates axes in the position specified by p. MATLAB® numbers subplot positions by row. The first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on. If axes exist in the specified position, then this command makes the axes the current axes.
8) Sin wave
Y = sin(X)

Y = sin(X) returns the sine of the elements of X. The sin function operates elementwise on arrays. The function accepts both real in [-1, 1] and complex inputs.

Code:

1) Sin wave with Titling, Labelling, Text, and text in Greek Letters

```
%sin wave of given frequency
f0=10; %frequency of wave
T=1; %Total duration
Fs=1000; %sampling freq
t=[0:1/Fs:T];
s1= sin(2*pi*f0*t);
figure;
plot(t,s1,'linewidth',1.5); %plot the sin
title('Sinusoidal Signal') %titling
xlabel('time(s)')
ylabel('Amplitude') %labelling
text(0.5,0.5,'sin(2\pif_0t)') %adding texts
```

2) Cos wave with Titling, Labelling, Text and text in Greek Letters

```
%cos wave of given frequency
f01=10; %frequency of wave
T1=1; %Total duration
Fs1=1000; %sampling freq
t1=[0:1/Fs:T];
s11= cos(2*pi*f0*t);
figure;
plot(t1,s11,'linewidth',1.5); %plot the sin
title(' Cosine Signal') %titling
xlabel('time(s)')
ylabel('Amplitude') %labelling
text(0.5,0.5,'cos(2\pif_0t)') %adding texts
```

3) Multiple plots with Titling, Labelling, Text, legends, and text in Greek Letters

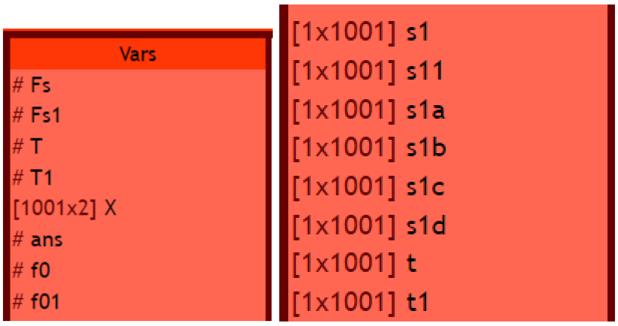
```
%multiple plots
X=[s1',s11'];
figure;
plot(X)
xlabel('time(s)')
ylabel('Amplitude') %labelling
legend('sin(2\pif_0t)','cos(2\pif_0t)')
text(200,0.5,'sin(2\pif_0t)') %adding texts
text(200,0.6,'cos(2\pif_0t)') %adding texts
```

4) Subplots with Titling, Labelling, Text, legends, and text in Greek Letters

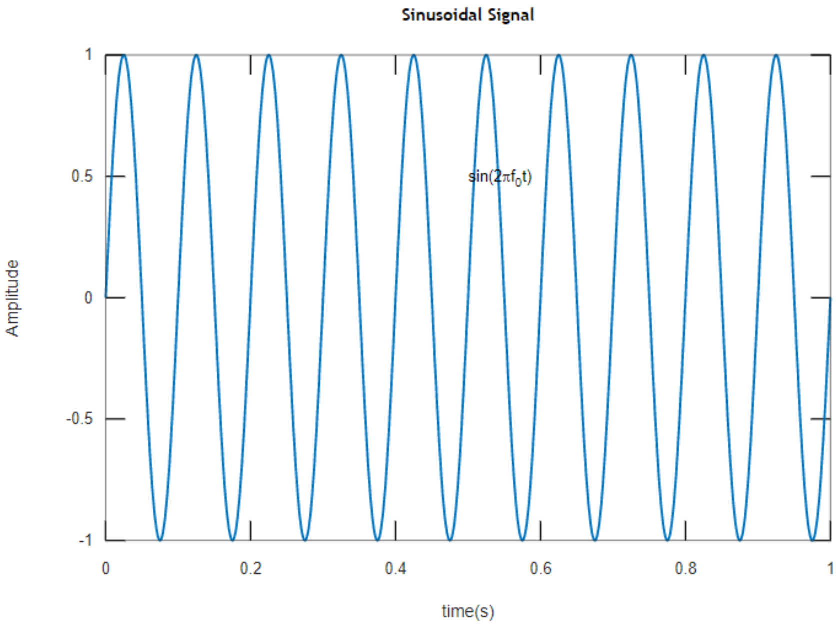
```
%changing frequency and making subplot
figure;
s1a = sin(2*pi*f0*2*t);
s1b = sin(2*pi*f0*4*t);
s1c = sin(2*pi*f0*0.25*t);
s1d = sin(2*pi*f0*0.0625*t);
subplot(2,2,1)
plot(t,s1a)
xlabel('time(s)')
ylabel('Amplitude') %labelling
subplot(2,2,2)
plot(t,s1b)
xlabel('time(s)')
ylabel('Amplitude') %labelling
subplot(2,2,3)
plot(t,s1c)
xlabel('time(s)')
ylabel('Amplitude') %labelling
subplot(2,2,4)
plot(t,s1d)
xlabel('time(s)')
ylabel('Amplitude') %labelling
```

Results:

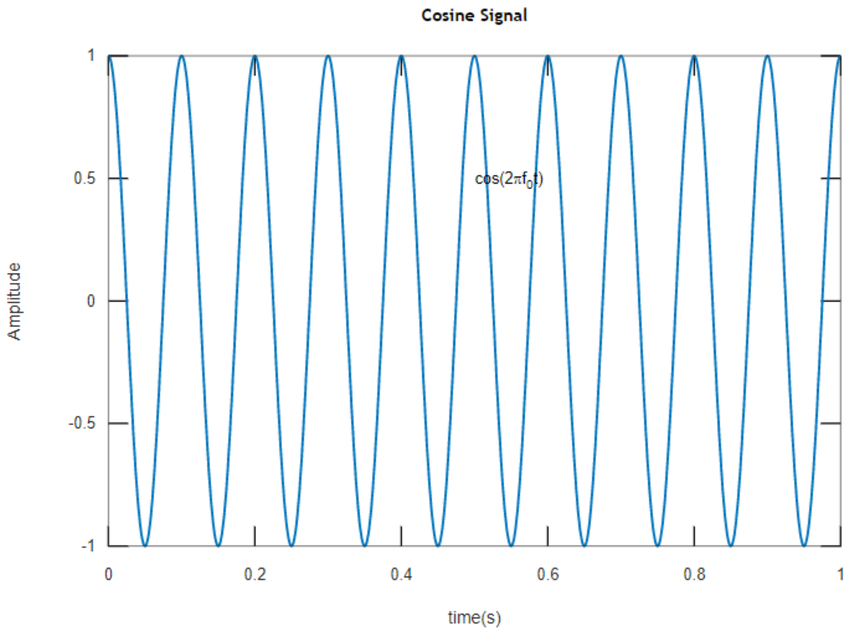
Total variables:



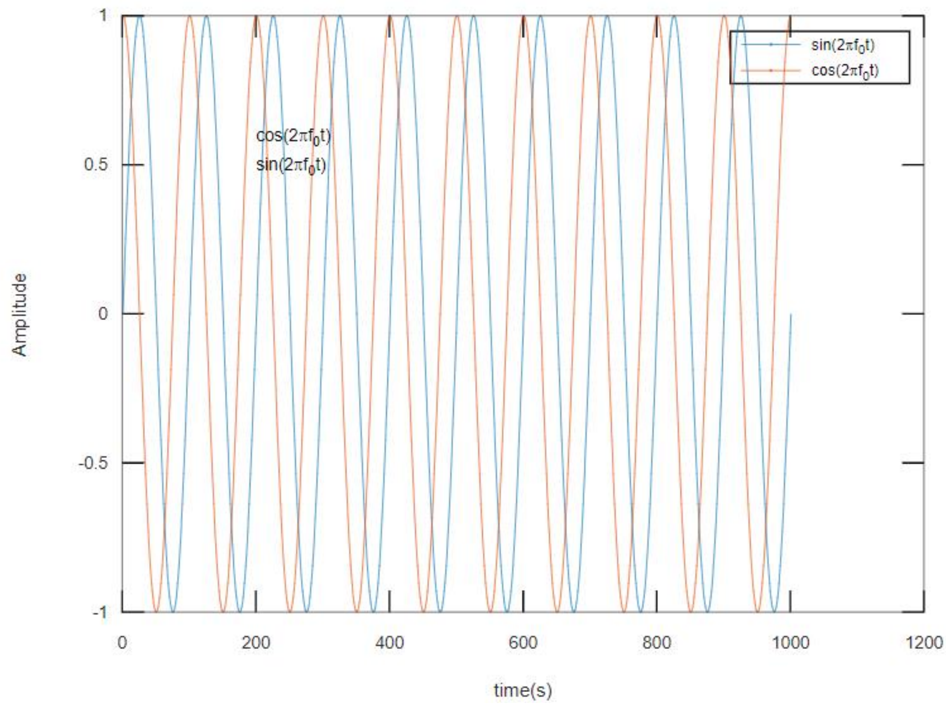
Sin wave with Titling, Labelling, Text, and text in Greek Letters:



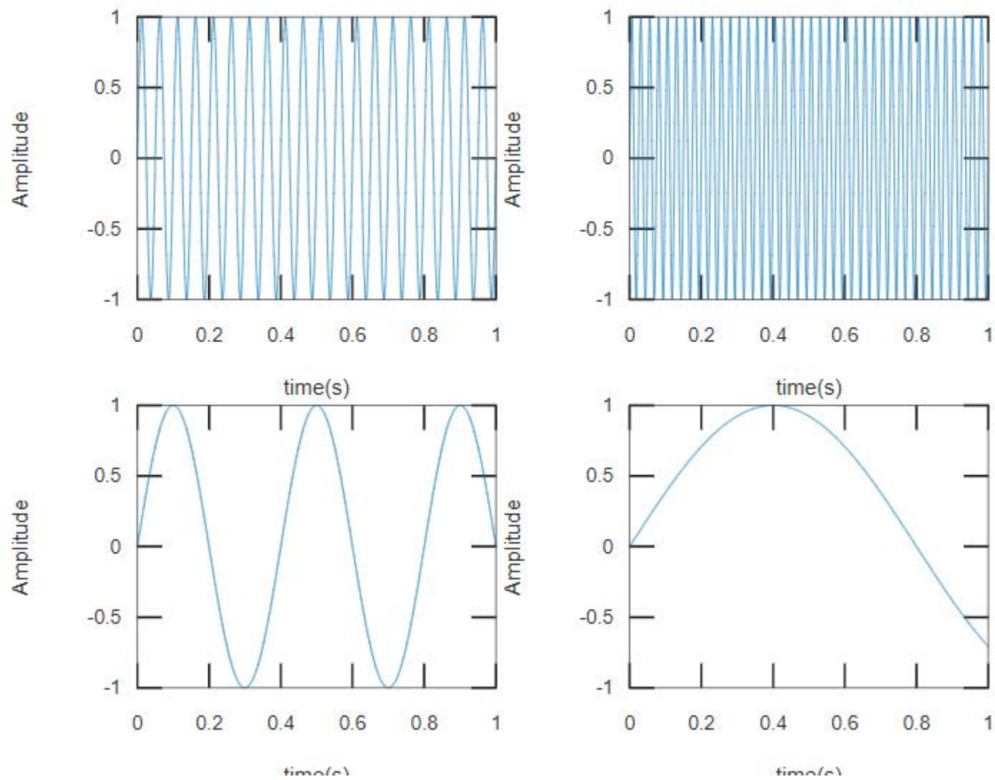
Cos wave with Titling, Labelling, Text, and text in Greek Letters:



Multiple plots with Titling, Labelling, Text, legends, and text in Greek Letters:



Subplots with Titling, Labelling, Text, legends, and text in Greek Letters:



Conclusion:

From this experiment we learn how to generate a Sinusoidal Signal of a given frequency with Titling, Labelling, Adding Text, Adding Legends, Printing Text in Greek Letters, Plotting as Multiple and Subplot and time scale the generated signal for different values. E.g. 2X, 4X, 0.25X, 0.0625X.

Experiment: 7

Aim: Solving Ordinary Differential Equation using Built-in Functions and plot.

1. First Order ordinary differential equation
2. Second Order ordinary differential equation
3. Third Order ordinary differential equation

Description:

1. ODE function

   I. [t,y] = ode45(odefun,tspan,y0)

      [t,y] = ode45(odefun,tspan,y0), where tspan = [t0 tf], integrates the system of differential equations $y'=f(t,y)$ from t0 to tf with initial conditions y0. Each row in the solution array y corresponds to a value returned in column vector t.

   II. [t,y] = ode45(odefun,tspan,y0,options)

      [t,y] = ode45(odefun,tspan,y0,options) also uses the integration settings defined by options, which is an argument created using the odeset function. For example, use the AbsTol and RelTol options to specify absolute and relative error tolerances, or the Mass option to provide a mass matrix.

   III. [t,y,te,ye,ie] = ode45(odefun,tspan,y0,options)

      [t,y,te,ye,ie] = ode45(odefun,tspan,y0,options) additionally finds where functions of (t,y), called event functions, are zero. In the output, te is the time of the event, ye is the solution at the time of the event, and ie is the index of the triggered event.

   IV. sol = ode45()

      sol = ode45(___) returns a structure that you can use with deval to evaluate the solution at any point on the interval [t0 tf]. You can use any of the input argument combinations in previous syntaxes.

2. Multiple plots in same figure:
   Create a line plot of both sets of data using plot function.

Code:

1. First order ODE

```
%first-order ODE
diffeq1 = @(t,y)[5];
tspan= [0 2];
y0=2;
[t,y]=ode45(diffeq1,tspan,y0);
figure;
plot(t,y,'linewidth',2);
xlabel('time');
ylabel('y')
title('First-order ODE sol of dy/dt=5')
```

2. Second order ODE

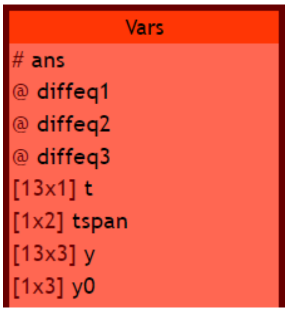
```
%second-order ODE
diffeq2 = @(t,y)[4;0];
tspan= [0 2];
y0=[3 4];
[t,y]=ode45(diffeq2,tspan,y0);
figure;
plot(t,y,'linewidth',2);
xlabel('time');
ylabel('y')
title('Second order ODE sol of d2y/dt2=0')
legend('y_1','y_2');
```

3. Third order ODE

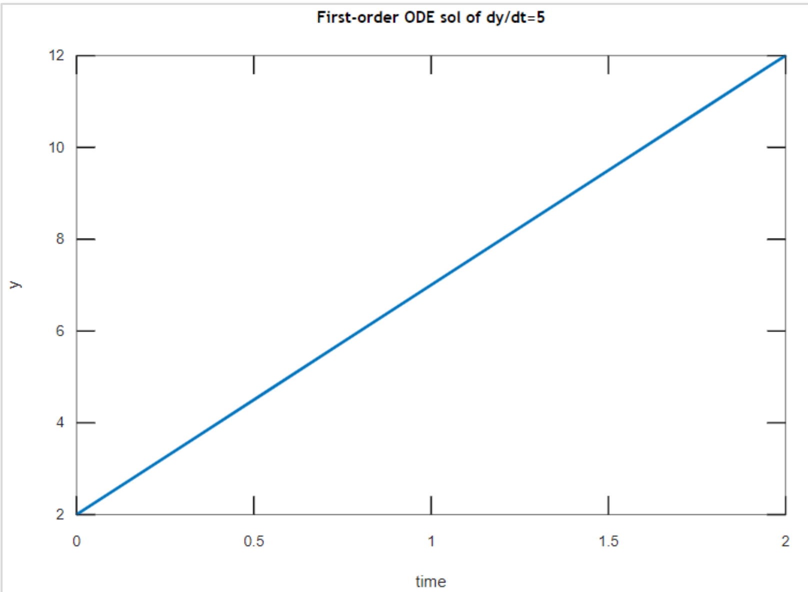
```
%third-order ODE
diffeq3 = @(t,y)[6*t+4;6;0];
tspan= [0 2];
y0=[3 4 6];
[t,y]=ode45(diffeq3,tspan,y0);
figure;
plot(t,y,'linewidth',2);
xlabel('time');
ylabel('y')
title('Third order ODE sol of d3y/dt3=0')
legend('y_1','y_2','y_3');
```

Results:

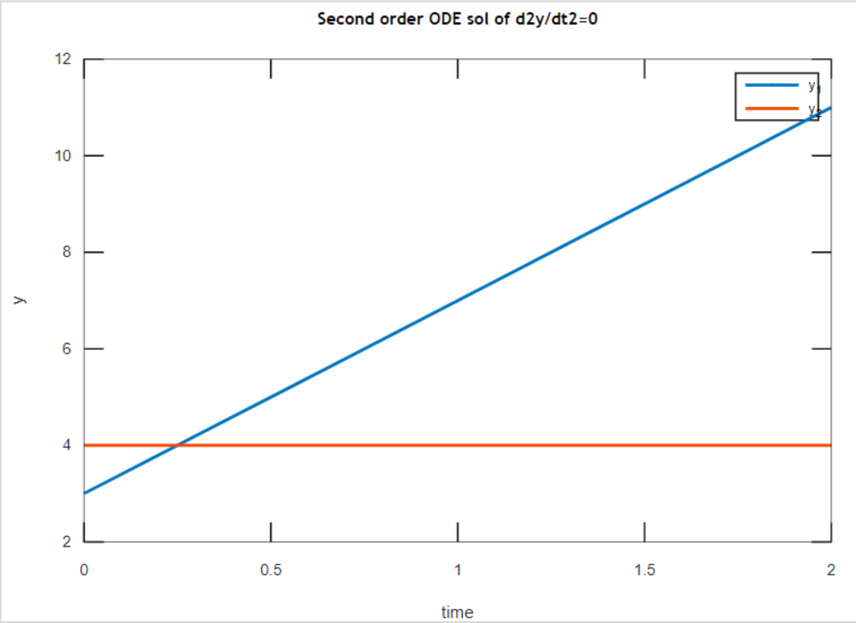
Total variables:



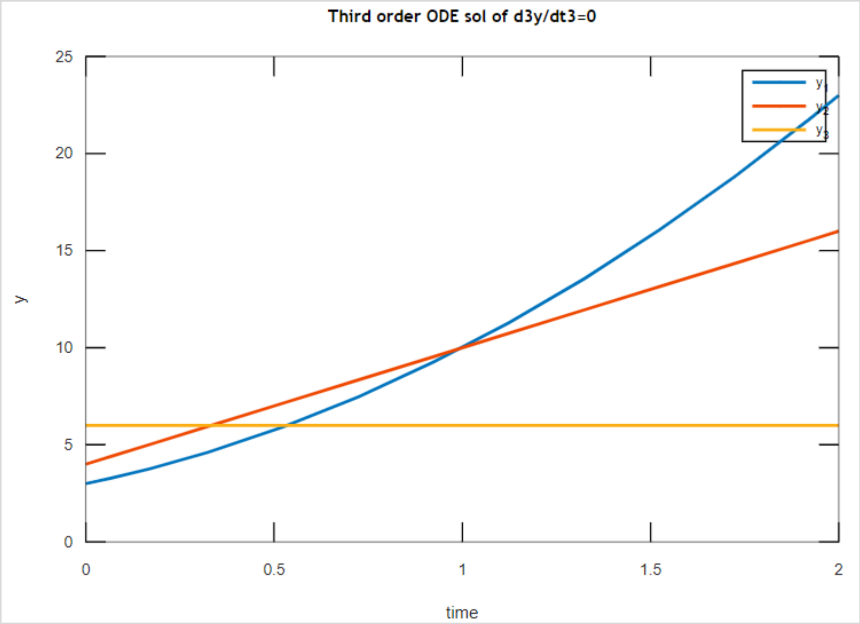
First order ODE:



Second order ODE:



Third order ODE:



Conclusion:

From this experiment we learn how to solve ordinary differential Equation using Built-in Functions and plot for first Order ordinary differential equation, Second Order ordinary differential equation and Third Order ordinary differential equation.

Experiment: 8

Aim: Writing brief Scripts starting each Script with a request for input (using input) to Evaluate the function h(T) using if-else statement, where, h(T) = (T – 10) for 0 < T < 100 h(T) = (0.45 T + 900) for T > 100.

Description:

1) Input

   x = input(prompt)

   str = input(prompt,'s')

   I. x = input(prompt) displays the text in prompt and waits for the user to input a value and press the **Return** key. The user can enter expressions, like pi/4 or rand(3), and can use variables in the workspace.

   II. If the user presses the **Return** key without entering anything, then input returns an empty matrix.

2) if, elseif, else

   if *expression*

   statements

   elseif *expression*

   statements

   else

   statements

   end

   I. if *expression*, *statements*, end evaluates an expression, and executes a group of statements when the expression is true. An expression is true when its result is nonempty and contains only nonzero elements (logical or real numeric). Otherwise, the expression is false.

   II. The elseif and else blocks are optional. The statements execute only if previous expressions in the if...end block are false. An if block can include multiple elseif blocks.

3) While

   while *expression*

   statements

   end

I. while *expression*, *statements*, end evaluates an expression, and repeats the execution of a group of statements in a loop while the expression is true. An expression is true when its result is nonempty and contains only nonzero elements (logical or real numeric). Otherwise, the expression is false.

4) Error

I. error(msg)

error(msg) throws an error and displays an error message.

II. error(msg,A1,...,An)

error(msg,A1,...,An) displays an error message that contains formatting conversion characters, such as those used with the MATLAB® sprintf function. Each conversion character in msg is converted to one of the values A1,...,An.

III. error(errID,___)

error(errID,___) includes an error identifier on the exception. The identifier enables you to distinguish errors and to control what happens when MATLAB encounters the errors. You can include any of the input arguments in the previous syntaxes.

IV. error(errorStruct)

error(errorStruct) throws an error using the fields in a scalar structure.

V. error(correction,___)

error(correction,___) provides a suggested fix for the exception. You can include any of the input arguments in the previous syntaxes.

Code:

1) Input variables:

```
clear all;
clc;
close all;

%input variables
T=input('Enter the value of T (T>0): ');
```

2) If-end

```
%if-end
if 100>T&&T>0
    display('1)h(T)=');
    h=T-10
end
```

3) If-else-end

```
if 100>T && T>0
   display('h(T)=');
    h=T-10
else
    display('h(T)=');
    h=0.45*T+900
end
```

4) If-elseif-else-end

```
%if-elseif-else-end
if 100>T&&T>0
    display('4)h(T)=');
    h=T-10
elseif T>100
    display('5)h(T)=');
    h=0.45*T+900
else
    error('Entered value T<0')
end
```

5) While loop

```
%while loop
flag='y'
while flag=='y'
    flag=input('Do you want to continue(y/n): ','s');
end
```

6) While-if-elseif-else-end

```
%while loop
flag='y'
while flag=='y'
    T=input('Enter the value of T (T>0): ');
        %if-elseif-else-end
        if 100>T&&T>0
            display('6)h(T)=');
            h=T-10
        elseif T>100
            display('7)h(T)=');
            h=0.45*T+900
        else
            error('Entered value T<0')
        end
    flag=input('Do you want to continue(y/n): ','s');
end
```

Results:

Total variables:

Vars
T
ans
(abc) flag
h

Input variables:

```
Enter the value of T (T>0): > 6
```

If-end :

```
Enter the value of T (T>0): > 5
1)h(T)=
h = -5
```

If-else-end :

```
Enter the value of T (T>0): > 4
2)h(T)=
h = -6
```

```
Enter the value of T (T>0): > 2000
3)h(T)=
h = 1800
```

If-elseif-else-end:

```
Enter the value of T (T>0): > 3
4)h(T)=
h = -7
```

```
Enter the value of T (T>0): > 500
5)h(T)=
h = 1125
```

```
Enter the value of T (T>0): > -500
error: Entered value T<0
error: called from
    exp_8 at line 16 column 5
```

While loop:

```
flag = y
Enter the value of T (T>0): > 5
Do you want to continue(y/n): > y
Enter the value of T (T>0): > 7
Do you want to continue(y/n): > n
```

While-if-elseif-else-end:

```
Enter the value of T (T>0): flag = y
> 6
6)h(T)=
h = -4
Do you want to continue(y/n): > y
Enter the value of T (T>0): > 600
7)h(T)=
h = 1170
Do you want to continue(y/n): > y
Enter the value of T (T>0): > -9
error: Entered value T<0
error: called from
    exp_8 at line 20 column 13
```

Conclusion:

From this experiment we learn how to write brief Scripts starting each Script with a request for input (using input) to Evaluate the function h(T) using if-else statement, where, $h(T) = (T - 10)$ for $0 < T < 100$ $h(T) = (0.45 T + 900)$ for $T > 100$.

Experiment: 9

Aim: Generating a Square Wave from sum of Sine Waves of certain Amplitude and Frequencies.

Description:

1. Y = sin(X)
   Y = sin(X) returns the sine of the elements of X. The sin function operates elementwise on arrays. The function accepts both real in the interval [-1, 1] and complex inputs.

2. Sum:

   S = sum(A)

   IV. S = sum(A) returns the sum of the elements of A along the first array dimension whose size does not equal 1.

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

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

3. Multiple plots in same figure:
   Create a line plot of both sets of data using plot function.
4. Labels
   I. xlabel(txt)
   xlabel(txt) labels the *x*-axis of the current axes or standalone visualization. Reissuing the xlabel command replaces the old label with the new label.
   II. ylabel(txt)
   ylabel(txt) labels the *y*-axis of the current axes or standalone visualization. Reissuing the ylabel command causes the new label to replace the old label.
5. Title
   title(titletext)
   title(titletext) adds the specified title to the current axes or standalone visualization. Reissuing the title command causes the new title to replace the old title.
6. Legend
   legend(label1,...,labelN)
   legend(label1,...,labelN) sets the legend labels. Specify the labels as a list of character vectors or strings, such as legend('Jan','Feb','Mar').
7. Text
   text(x,y,txt)

text(x,y,txt) adds a text description to one or more data points in the current axes using the text specified by txt. To add text to one point, specify x and y as scalars. To add text to multiple points, specify x and y as vectors with equal length.

8. For loop

for index = values
  statements
end

I. for *index* = *values*, *statements*, end executes a group of statements in a loop for a specified number of times. *values* has one of the following forms:

II. *initVal*:*endVal* — Increment the *index* variable from *initVal* to *endVal* by 1, and repeat execution of *statements* until *index* is greater than *endVal*.

III. *initVal*:*step*:*endVal* — Increment *index* by the value *step* on each iteration, or decrements *index* when *step* is negative.

IV. *valArray* — Create a column vector, *index*, from subsequent columns of array *valArray* on each iteration. For example, on the first iteration, *index* = *valArray*(:,1). The loop executes a maximum of *n* times, where *n* is the number of columns of *valArray*, given by numel(*valArray*(1,:)). The input *valArray* can be of any MATLAB® data type, including a character vector, cell array, or struct.

Code:

1. For loop

```
clear all;
clc;
%for loop
sumA=0;
A=[1,3,8,-2];
for k=1:length(A)
    sumA+=A(k)
end
```

2. Square wave generation

```
%square-wave generation-1
f0=10;
T=1;
fs=1000;
t=[0:1/fs:T];
s1=sin(2*pi*f0.*t);
sq=(4/pi)*sin(2*pi*f0.*t);
for k=3:2:10
    sq+=(4/(k*pi))*sin(2*pi*k*f0.*t);
end
figure;
plot(t,s1,t,sq,'linewidth',1.5)
xlabel('time (t)')
ylabel("waves")
title('square wave vs sine wave')
legend('sin(2*pi*f_0.*t)','(4/pi)*sin(2*pi*f0.*t);')
text(0.8,0.8,'sin(2*pi*f_0.*t)','fontsize',14)
text(0.8,0.6,'sin(2\omega t)','fontsize',14)
```

```
%square-wave generation-2
f0=10;
T=1;
fs=1000;
t=[0:1/fs:T];
s1=sin(2*pi*f0.*t);
sq=(4/pi)*sin(2*pi*f0.*t);
for k=3:2:10
    sq+=(4/(k*pi))*sin(2*pi*k*f0.*t);
end
figure;
plot(t,s1,t,sq,'linewidth',1.5)
xlabel('time (t)')
ylabel("waves")
title('square wave vs sine wave')
legend('sin(2*pi*f_0.*t)','(4/pi)*sin(2*pi*f0.*t);')
text(0.8,0.8,'sin(2*pi*f_0.*t)','fontsize',14)
text(0.8,0.6,'sin(2\omega t)','fontsize',14)
```

```
%square-wave generation-3
f0=10;
T=1;
fs=1000;
t=[0:1/fs:T];
s1=sin(2*pi*f0.*t);
sq=(4/pi)*sin(2*pi*f0.*t);
for k=3:2:100
    sq+=(4/(k*pi))*sin(2*pi*k*f0.*t);
end
figure;
plot(t,s1,t,sq,'linewidth',1.5)
xlabel('time (t)')
ylabel("waves")
title('square wave vs sine wave')
legend('sin(2*pi*f_0.*t)','(4/pi)*sin(2*pi*f0.*t);')
text(0.8,0.8,'sin(2*pi*f_0.*t)','fontsize',14)
text(0.8,0.6,'sin(2\omega t)','fontsize',14)
```

Results:

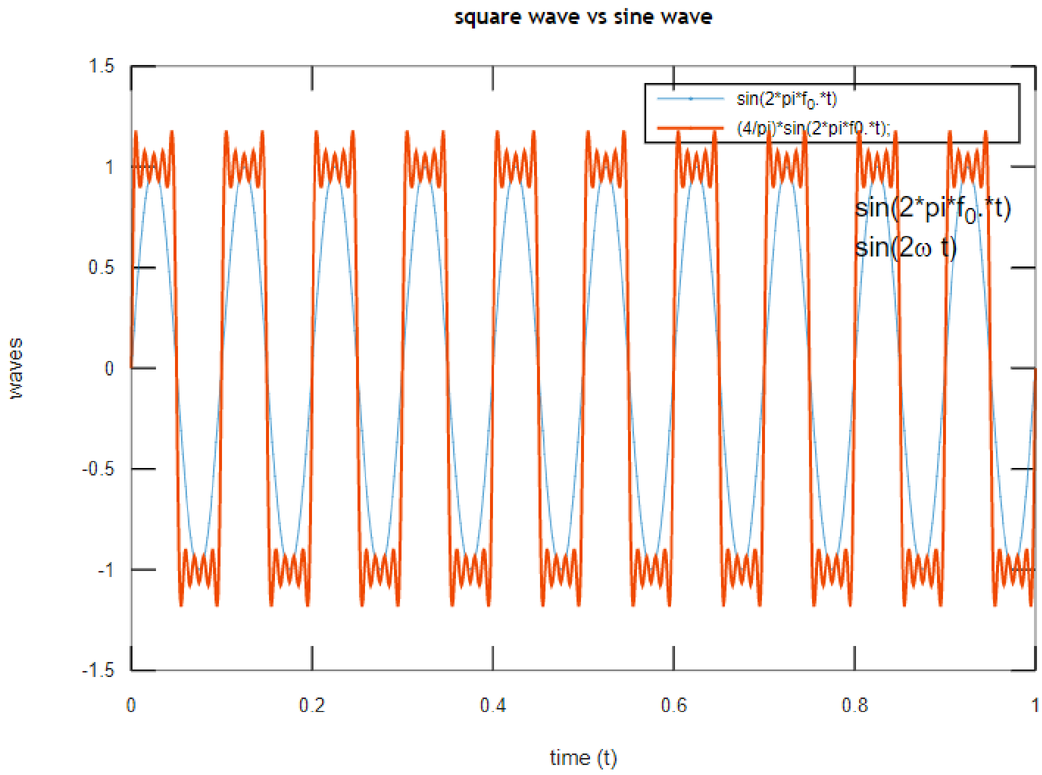
Total variables:

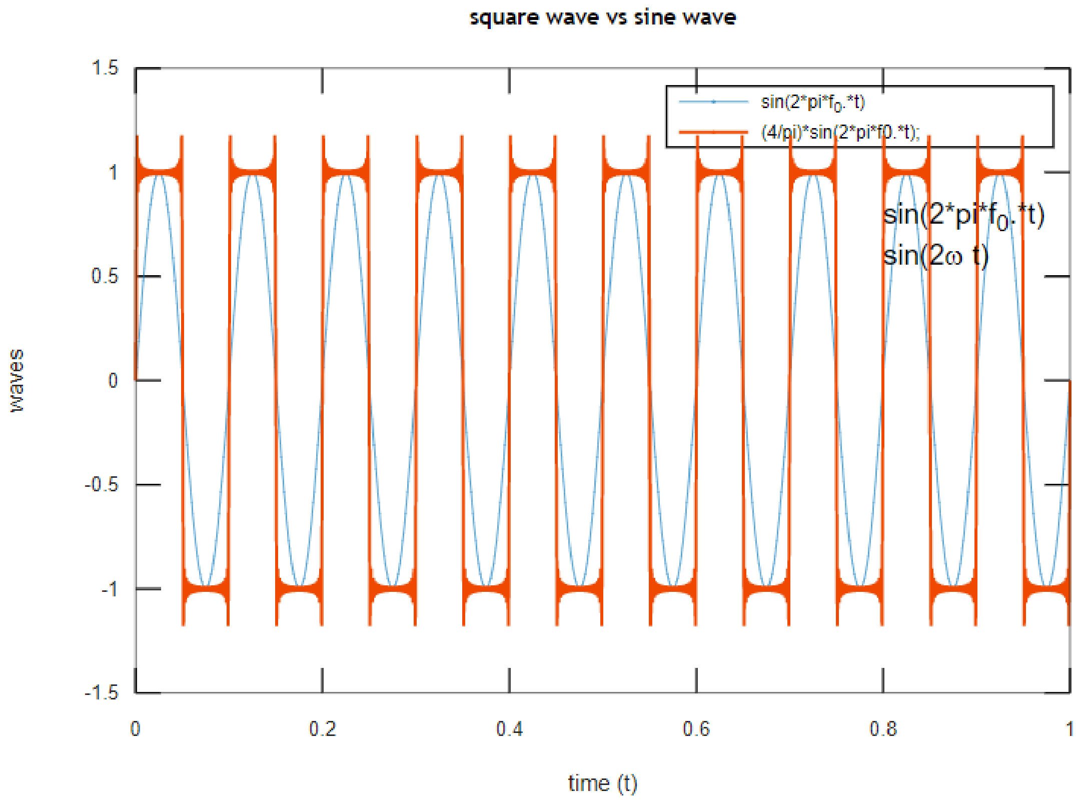
Vars
[1x4] A
T
ans
f0
fs
k
[1x1001] s1
[1x1001] sq
sumA
[1x1001] t

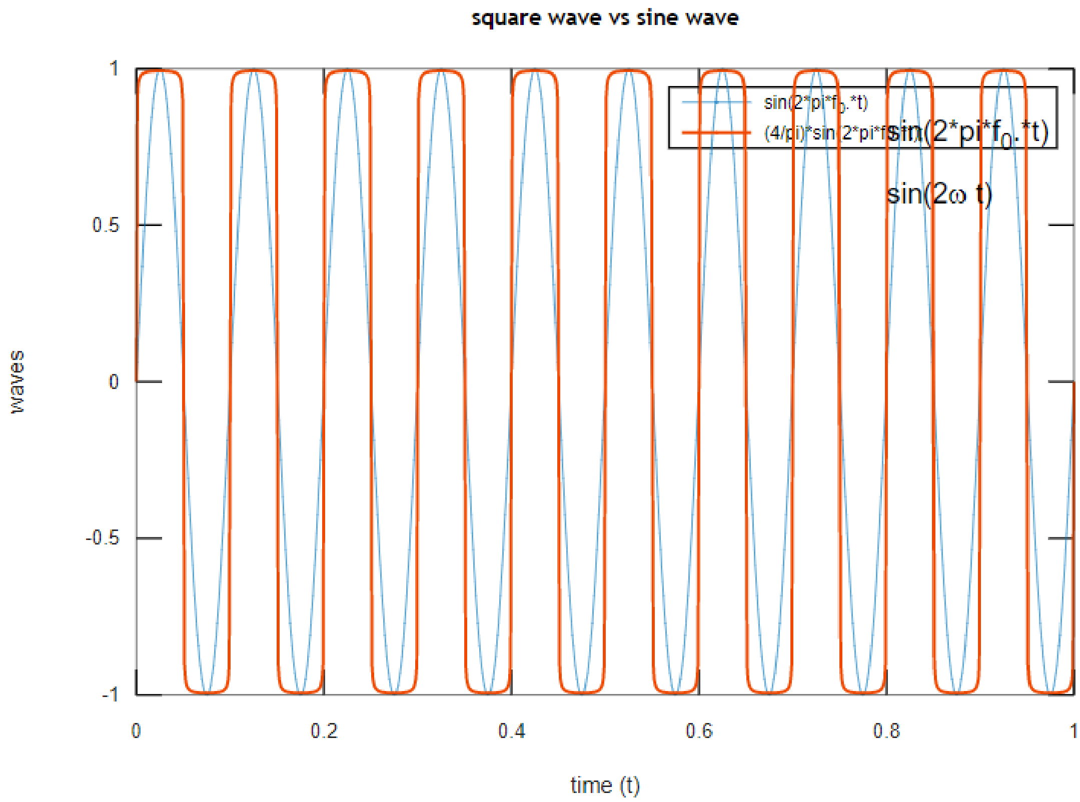
For loop :

```
sumA = 1
sumA = 4
sumA = 12
sumA = 10
ans = -305.27
```

Square wave generation :







Conclusion:

From this experiment we learn how to generate a Square Wave from sum of Sine Waves of certain Amplitude and Frequencies.

Experiment: 10

Aim: Basic 2D and 3D plots: parametric space curve, polygons with vertices, 3D contour lines and pie and bar charts.

Description:

1. Ezplot() :
   I. ezplot(fun)

      ezplot(fun) plots the expression fun(x) over the default domain -2π < x < 2π, where fun(x) is an explicit function of only x. fun can be a function handle, a character vector, or a string.

   II. ezplot(fun,[xmin,xmax])
      ezplot(fun,[xmin,xmax]) plots fun(x) over the domain: xmin < x < xmax.
2. Ezplot3() :
   I. ezplot3(funx,funy,funz)

      ezplot3(funx,funy,funz) plots the spatial curve funx(t), funy(t), and funz(t) over the default domain 0 < t < 2π. funx, funy, and funz can be function handles, character vectors, or strings (see the Tips section).

   II. ezplot3(funx,funy,funz,[tmin,tmax])
      ezplot3(funx,funy,funz,[tmin,tmax]) plots the curve funx(t), funy(t), and funz(t) over the domain tmin < t < tmax.
3. Xlim :
   xlim(limits)
   xlim(limits) sets the *x*-axis limits for the current axes or chart. Specify limits as a two-element vector of the form [xmin xmax], where xmax is greater than xmin.
4. Ylim :
   ylim(limits)
   ylim(limits) sets the *y*-axis limits for the current axes or chart. Specify limits as a two-element vector of the form [ymin ymax], where ymax is greater than ymin.
5. Grid :
   I. grid on
      grid on displays the major grid lines for the current axes or chart returned by the gca command. Major grid lines extend from each tick mark.
   II. grid off
      grid off removes all grid lines from the current axes or chart.
6. DrawPolygon :
   roi = drawpolygon creates a Polygon ROI object and enables interactive drawing of a polygonal region-of-interest (ROI) on the current axes.

I. roi = drawpolygon

II. roi = drawpolygon(ax)

roi = drawpolygon(ax) creates the ROI on the axes specified by ax.

7. Meshgrid :

I. [X,Y] = meshgrid(x,y)

It returns 2-D grid coordinates based on the coordinates contained in vectors x and y. X is a matrix where each row is a copy of x, and Y is a matrix where each column is a copy of y. The grid represented by the coordinates X and Y has length(y) rows and length(x) columns.

II. [X,Y] = meshgrid(x)

It is the same as [X,Y] = meshgrid(x,x), returning square grid coordinates with grid size length(x)-by-length(x).

III. [X,Y,Z] = meshgrid(x,y,z)

It returns 3-D grid coordinates defined by the vectors x, y, and z. The grid represented by X, Y, and Z has size length(y)-by-length(x)-by-length(z).

IV. [X,Y,Z] = meshgrid(x)

It is the same as [X,Y,Z] = meshgrid(x,x,x), returning 3-D grid coordinates with grid size length(x)-by-length(x)-by-length(x).

8. Surfc :

I. surfc(X,Y,Z)

It creates a three-dimensional surface plot with a contour plot underneath. A surface plot is a three-dimensional surface that has solid edge colors and solid face colors. The function plots the values in matrix Z as heights above a grid in the *x*-*y* plane defined by X and Y. The color of the surface varies according to the heights specified by Z.

II. surfc(X,Y,Z,C)

It additionally specifies the surface color.

III. surfc(Z)

It creates a surface and contour plot and uses the column and row indices of the elements in Z as the *x*- and *y* -coordinates.

9. Bar :

Bar graph

I. bar(y)

bar(y) creates a bar graph with one bar for each element in y. If y is an *m*-by-*n* matrix, then bar creates *m* groups of *n* bars.

II. bar(x,y)

bar(x,y) draws the bars at the locations specified by x.

10. Pie :

Pie chart

pie(X) draws a pie chart using the data in X. Each slice of the pie chart represents an element in X.

- If sum(X) ≤ 1, then the values in X directly specify the areas of the pie slices. pie draws only a partial pie if sum(X) < 1.
- If sum(X) > 1, then pie normalizes the values by X/sum(X) to determine the area of each slice of the pie.
- If X is of data type categorical, the slices correspond to categories. The area of each slice is the number of elements in the category divided by the number of elements in X.

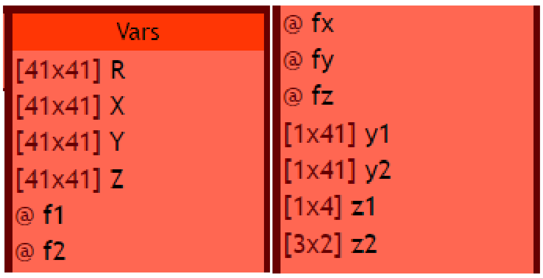
Code:

```
clc; clear all;
%%parametric curves
f1= @(x) x.^2-2;
figure;
ezplot(f1,[-2 2],100);
grid on;
xlabel('x');
xlim([-2 2]);
ylim([-2 2]);

%% cos(t) using ezplot
f2= @(t) cos(t);
figure;
ezplot(f2,[0 6*pi],100);
grid on;
xlabel('t');

%3-D parametric plots
fx=@(t) cos(t)
fy= @(t) sin(t)
fz= @(t) t;
figure;
ezplot3(fx,fy,fz,[0 6*pi],100);
grid on;
xlabel('cos(t)');
ylabel('sin(t)');
zlabel('(t)');

%polygon with vertices
figure;
drawPolygon([0,0; 1,0 ; 1,1 ;0,1]);
xlim([-0.1 1.1]);
ylim([-0.1 1.1]);
```

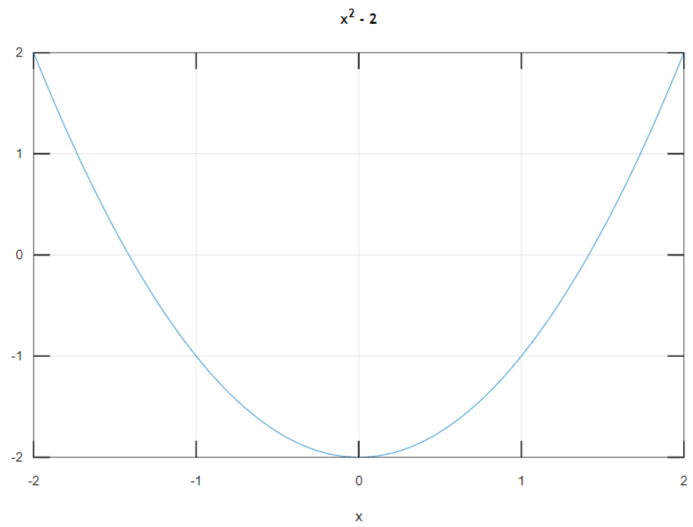
```
%polygon with vertices -triangle
figure;
drawPolygon([0,0; 1,0 ; 0.5,0.5]);
xlim([-0.1 1.1]);
ylim([-0.1 1.1]);

%surface plot
y1= -2*pi:pi/10:2*pi;
y2= -2*pi:pi/10:2*pi;
[X,Y]=meshgrid(y1,y2);
R = sqrt(X.^2+Y.^2);
Z = sin(R)./R;
figure;
surfc(Z);
xlabel('x');
ylabel('x');
zlabel('x');
%surface plot with contour
figure;
contour(Z);
colorbar;

%pie chart and pie bar
z1=[3,3,3,1];
figure; pie(z1);
figure; bar(z1);
z2=[1,2;4,1;2,0.5];
figure;bar(z2);
```

Results:

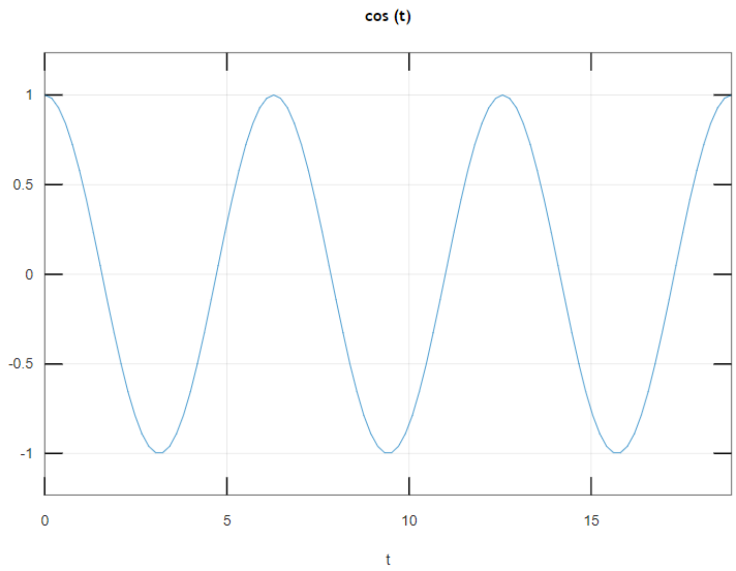
Total variables:



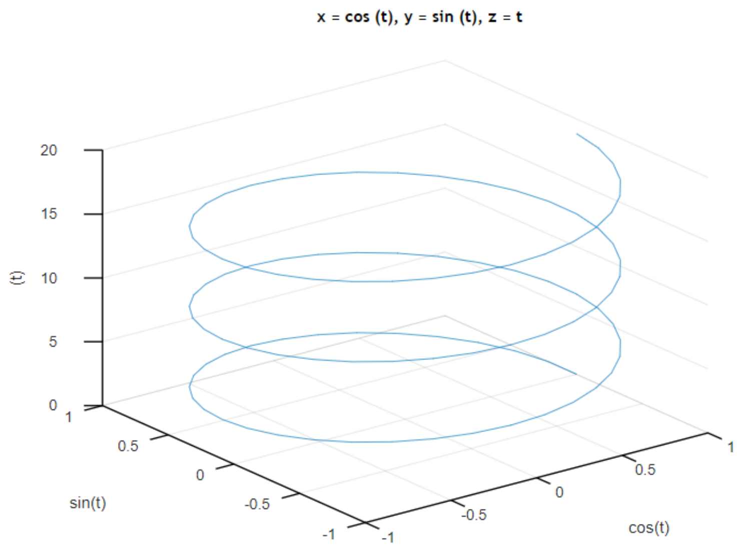
Parametric curves:



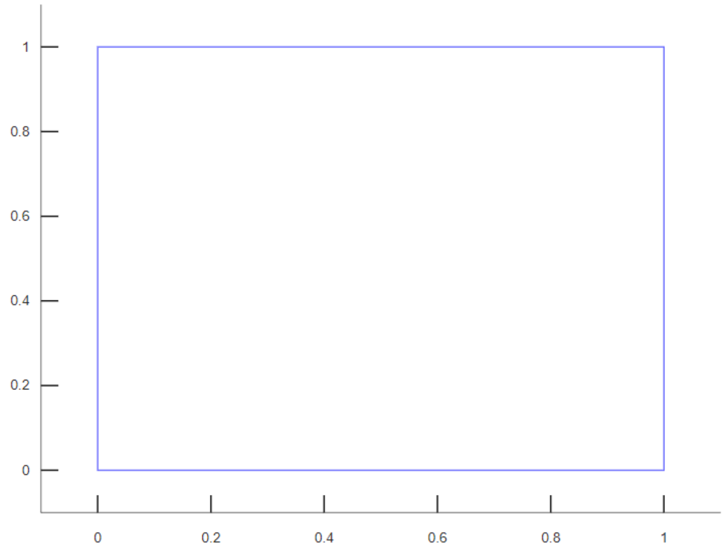
cos(t) using ezplot:



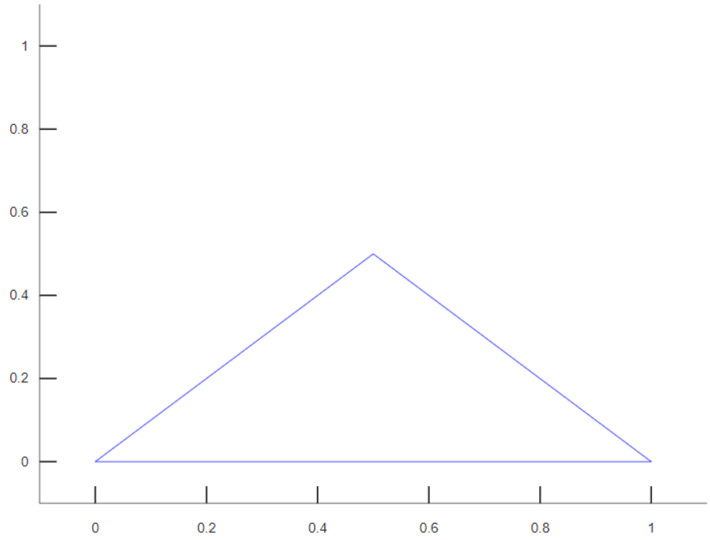
3-D parametric plots:



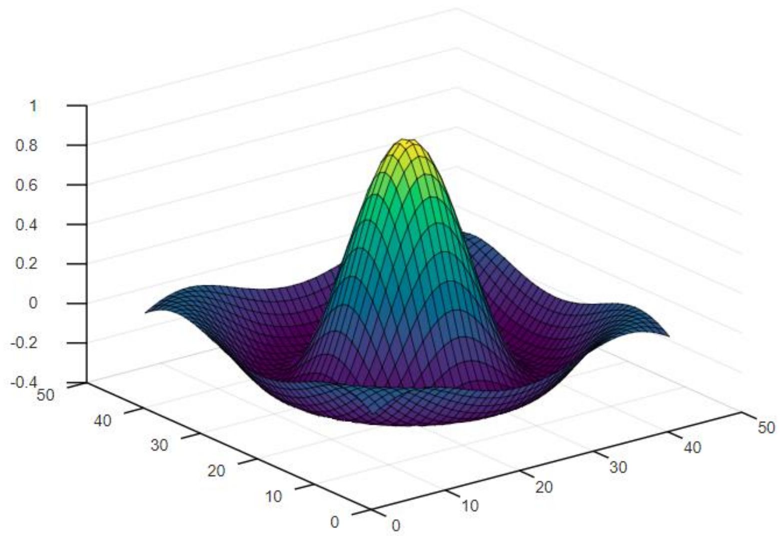
polygon with vertices:



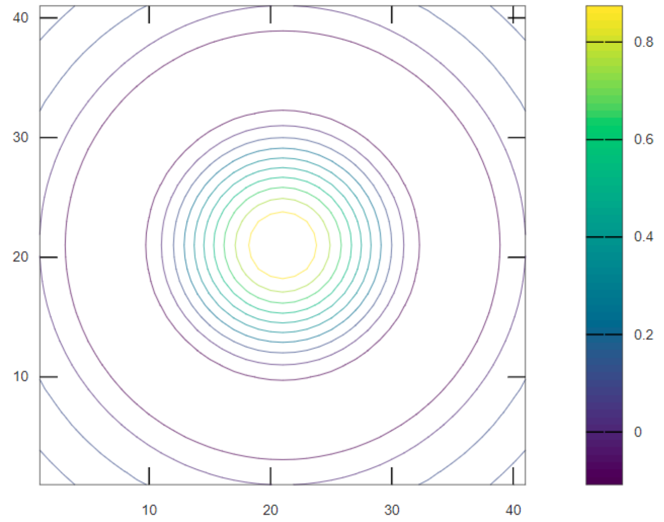
polygon with vertices -triangle:



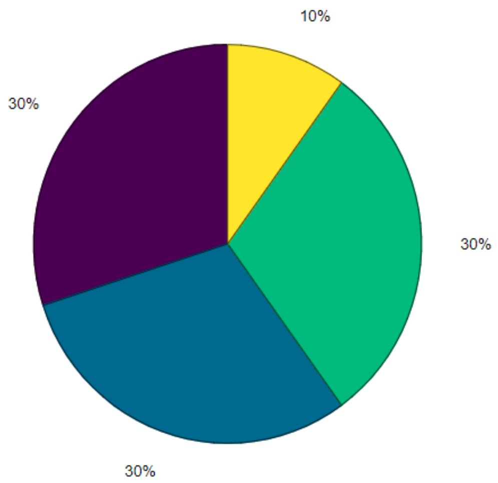
surface plot:



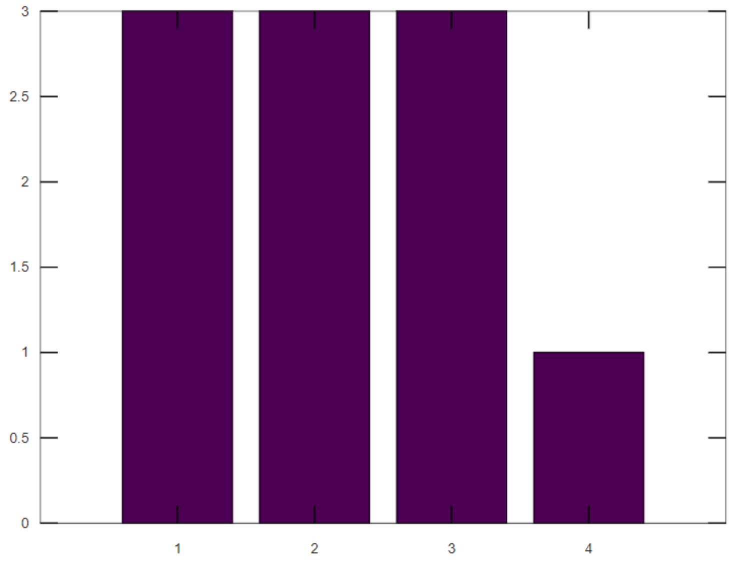
surface plot with contour:

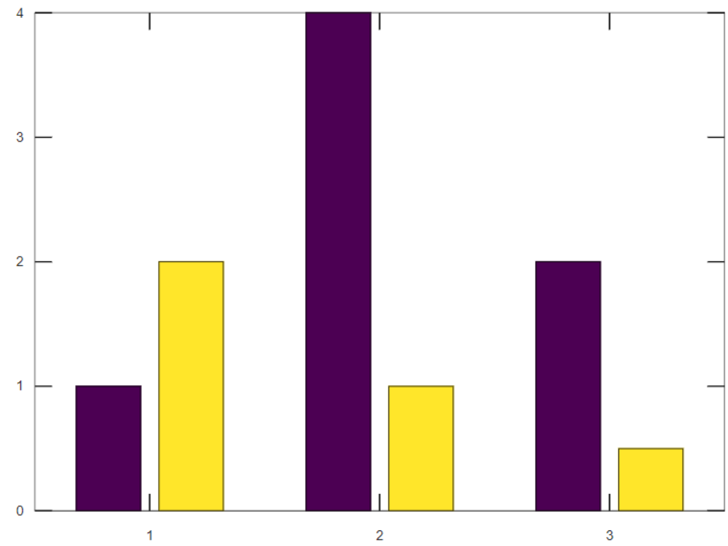


pie chart :



pie bar:





Conclusion:

From this experiment we learn about basic 2D and 3D plots: parametric space curve, polygons with vertices, 3D contour lines and pie and bar charts.