Lab File

On

BASIC SIMULATION

Submitted To

Amity University Uttar Pradesh

In partial fulfillment of the requirements for the award degree of
Bachelor of Technology
In
ARTIFICIAL INTELLIGENCE
By

SUNIDHI SINGH
Enrollment number A023119820032

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY
AMITY UNIVERSITY UTTAR PRADESH
NOIDA (U.P)

INDEX

	elements from 1 to j, inclusive. (C) Generating a Random Sequence using rand() / randn() functions and plot them.				
4.	Evaluating a given expression and rounding it to the nearest integer value using Round, Floor, Ceil and Fix functions; Also, generating and Plots of (A) Trigonometric Functions - sin(t),cos(t), tan(t), sec(t), cosec(t) and cot(t) for a given duration, ‘t’. (B) Logarithmic and other Functions – log(A), log10(A), Square root of A, Real nth root of A.				
5.	Creating a vector X with elements, Xn = (-1)n+1/(2n-1) and Adding up 100 elements of the vector, X; And, plotting the functions, x, x3, ex, exp(x2) over the interval 0 < x < 4 (by choosing appropriate mesh values for x to obtain smooth curves), on A Rectangular Plot .				
6.	Generating a Sinusoidal Signal of a given frequency with Titling, Labeling, Adding Text, Adding Legends, Printing Text in Greek Letters, Plotting as Multiple and Subplot. Time scale the generated signal for different values. E.g. 2X, 4X, 0.25X, 0.0625X.				

7.	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 $T > 100$.				
8.	Solving First, Second and third Order Ordinary Differential Equation using Built-in Functions and plot.				
9.	Generating a Square Wave from sum of Sine Waves of certain Amplitude and Frequencies.				
10.	Basic 2D and 3D plots: parametric space curve, polygons with vertices, 3D contour lines and pie and bar charts.				

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.

THEORY:

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include Math and computation, Algorithm development. Data acquisition, Modelling, simulation, and prototyping, Data analysis, exploration, and visualization, Scientific and engineering graphics, Application development, including graphical user interface building.

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a SUGGESTED PROGRAM: in a scalar non interactive language such as C or Fortran. The name MATLAB stands for **matrix laboratory**.

Matrix Addition and Subtraction is performed by merely adding or subtracting the matrix elements by their corresponding elements in the other matrix.

Consider two matrices A and B. If A is an m x n matrix and B is an n x p matrix, they could be multiplied together to produce an m x n matrix C.
Matrix multiplication is possible only if the number of columns n in A is equal to the number of rows n in B. In matrix multiplication, the elements of the rows in the first matrix are multiplied with corresponding columns in the second matrix. Each element in the (i, j) position, in the resulting matrix C, is the summation of the products of elements in i row of first matrix with the corresponding element in the j column of the second matrix. In MATLAB, matrix multiplication is performed by using the *operator.

The inverse of a matrix does not always exist.
If the determinant of the matrix is zero, then the inverse does not exist and the matrix is singular.

Inverse of a matrix A is given by inv(A).

Transpose operation switches the rows and columns in a matrix. It is represented by a single quote (‘).
Rank function provides an estimate of the number of linearly independent rows or columns of a null matrix.k = rank(A) returns the number of singular values of A that are larger than the default tolerance, max(size(A))*eps(norm(A)).

k=rank(A, tol) returns the number of singular values of A that are larger than tol.

For 1- Dimensional array:

CODE:

```
A=[1 2 3 4 5]
B=[7 6 8 9 0]
C=A+B
D=A-B
E=exp(A)
```

OUTPUT:

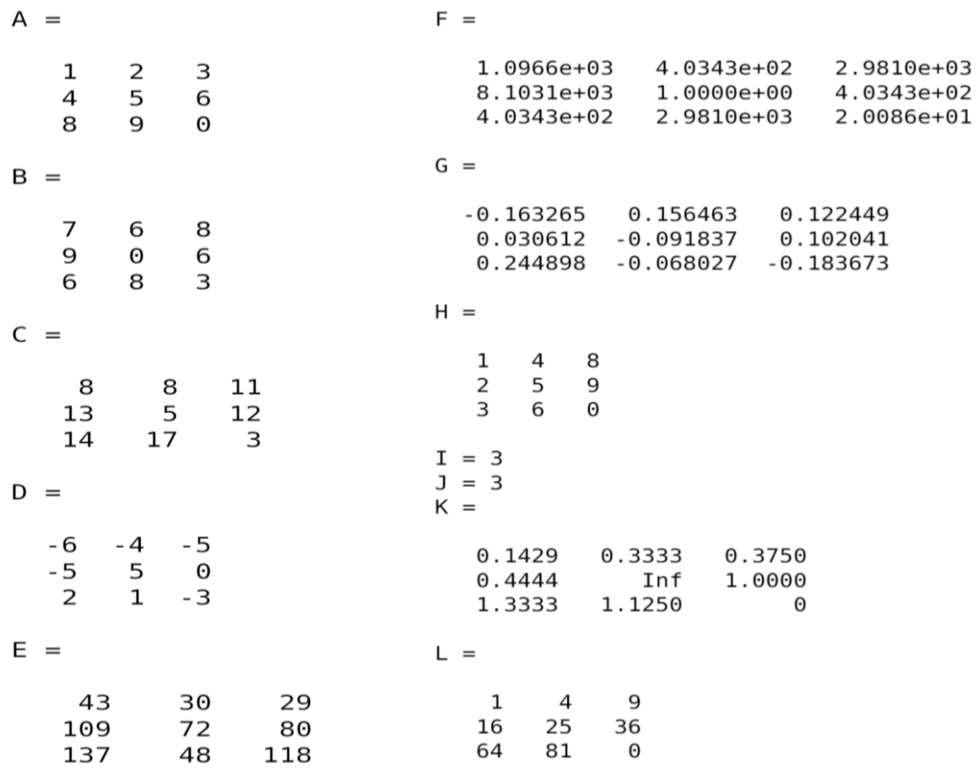
```
A =

   1   2   3   4   5

B =

   7   6   8   9   0

C =

    8    8   11   13     5

D =

  -6  -4  -5  -5   5

E =

     2.7183     7.3891    20.0855    54.5982   148.4132
```

For Multidimensional array(matrix)

CODE:

```
A=[1 2 3; 4 5 6; 8 9 0]
B=[7 6 8; 9 0 6; 6 8 3]
C=A+B
D=A-B
E=A*B
F=exp(B)
G=inv(B)
H=transpose(A)
I=rank(A)
J=rank(B)
K=A./B
L=A.^2
```

OUTPUT:



```
A =

   1   2   3
   4   5   6
   8   9   0

B =

   7   6   8
   9   0   6
   6   8   3

C =

    8    8   11
   13    5   12
   14   17    3

D =

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

E =

    43    30    29
   109    72    80
   137    48   118
```

```
F =

   1.0966e+03   4.0343e+02   2.9810e+03
   8.1031e+03   1.0000e+00   4.0343e+02
   4.0343e+02   2.9810e+03   2.0086e+01

G =

  -0.163265   0.156463   0.122449
   0.030612  -0.091837   0.102041
   0.244898  -0.068027  -0.183673

H =

   1   4   8
   2   5   9
   3   6   0

I = 3
J = 3
K =

   0.1429   0.3333   0.3750
   0.4444      Inf   1.0000
   1.3333   1.1250        0

L =

    1    4    9
   16   25   36
   64   81    0
```

CONCLUSION:

Arithmetic operations is successfully performed on 1 Dimensional array. Matrix operations are also successfully performed.

Criteria	Total Marks	Marks Obtained	Comments
Concept (a)	2		
Implementation (b)	2		
Performance (c)	2		
Total	6		

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

THEORY:

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. MATLAB is designed to operate primarily on whole matrices and arrays. Therefore, operators in MATLAB work both on scalar and non-scalar data.

MATLAB allows the following types of elementary operations −

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operations
5. Set Operations

Arithmetic Operators

MATLAB allows two different types of arithmetic operations:

- Matrix arithmetic operations
- Array arithmetic operations

Matrix arithmetic operations are same as defined in linear algebra.
Array operations are executed element by element, both on one-dimensional and multidimensional array.

The matrix operators and array operators are differentiated by the period (.) symbol.

However, as the addition and subtraction operation is same for matrices and arrays, the operator is same for both cases. The following table gives brief description of the operators:

Operator	Description
+	Addition or unary plus. A+B adds the values stored in variables A and B. A and B must have the same size unless one is a scalar. A scalar can be added to a matrix of any size.
-	Subtraction or unary minus. A-B subtracts the value of B from A. A and B must have the same size unless one is a scalar. A scalar can be subtracted from a matrix of any size.
*	Matrix multiplication. C = A*B is the linear algebraic product of the matrices A and B. More precisely, $C(i,j) = \sum_{k=1}^{n} A(i,k)B(k,j)$ For non-scalar A and B, the number of columns of A must be equal to the number of rows of B. A scalar can multiply a matrix of any size.
.*	Array multiplication. A.*B is the element-by-element product of the arrays A and B. A and B must have the same size, unless one of them is a scalar.
/	Slash or matrix right division. B/A is roughly the same as B*inv(A). More precisely, B/A = (A'\B')'.
./	Array right division. A./B is the matrix with elements A(i,j)/B(i,j). A and B must have the same size, unless one of them is a scalar.
\	Backslash or matrix left division. If A is a square matrix, A\B is roughly the same as inv(A)*B, except it is computed in a different way. If A is an n-by-n matrix and B is a column vector with n components, or a matrix with several such columns, then X = A\B is the solution to the equation *AX* = *B*. A warning message is displayed if A is badly scaled or nearly singular.
.\	Array left division. A.\B is the matrix with elements B(i,j)/A(i,j). A and B must have the same size, unless one of them is a scalar.
^	Matrix power. X^p is X to the power p, if p is a scalar. If p is an integer, the power is computed by repeated squaring. If the integer is negative, X is inverted first. For other values of p, the calculation involves eigenvalues and eigenvectors, such that if [V,D] = eig(X), then X^p = V*D.^p/V.
.^	Array power. A.^B is the matrix with elements A(i,j) to the B(i,j) power. A and B must have the same size, unless one of them is a scalar.

‘	Matrix transpose. A' is the linear algebraic transpose of A. For complex matrices, this is the complex conjugate transpose.
.’	Array transpose. A.' is the array transpose of A. For complex matrices, this does not involve conjugation.

Relational Operators:

Relational operators can also work on both scalar and non-scalar data. Relational operators for arrays perform element-by-element comparisons between two arrays and return a logical array of the same size, with elements set to logical 1 (true) where the relation is true and elements set to logical 0 (false) where it is not.
The following table shows the relational operators available in MATLAB:

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to

Logical Operators:
MATLAB offers two types of logical operators and functions:

- Element-wise − These operators operate on corresponding elements of logical arrays.
- Short-circuit − These operators operate on scalar and, logical expressions.

Element-wise logical operators operate element-by-element on logical arrays. The symbols &, |, and ~ are the logical array operators AND, OR, and NOT.
Short-circuit logical operators allow short-circuiting on logical operations. The symbols && and || are the logical short-circuit operators AND and OR.

Bitwise Operations:
Bitwise operators work on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows:

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1

1	1	1	1	0
1	0	0	1	1

Assume if A = 60; and B = 13; Now in binary format they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

MATLAB provides various functions for bit-wise operations like 'bitwise and', 'bitwise or' and 'bitwise not' operations, shift operation, etc.

The following table shows the commonly used bitwise operations:

Function	Purpose
bitand(a, b)	Bit-wise AND of integers *a* and *b*
bitcmp(a)	Bit-wise complement of *a*
bitget(a,pos)	Get bit at specified position *pos*, in the integer array *a*
bitor(a, b)	Bit-wise OR of integers *a* and *b*
bitset(a, pos)	Set bit at specific location *pos* of *a*
bitshift(a, k)	Returns *a* shifted to the left by *k* bits, equivalent to multiplying by 2^k. Negative values of k correspond to shifting bits right or dividing by $2^{\|k\|}$ and rounding to the nearest integer towards negative infinite. Any overflow bits are truncated.
bitxor(a, b)	Bit-wise XOR of integers *a* and *b*
swapbytes	Swap byte ordering

Set Operations:

MATLAB provides various functions for set operations, like union, intersection and testing for set membership, etc.

The following table shows some commonly used set operations:

Function	Description
intersect(A,B)	Set intersection of two arrays; returns the values common to both A and B. The values returned are in sorted order.

intersect(A,B,'rows')	Treats each row of A and each row of B as single entities and returns the rows common to both A and B. The rows of the returned matrix are in sorted order.
ismember(A,B)	Returns an array the same size as A, containing 1 (true) where the elements of A are found in B. Elsewhere, it returns 0 (false).
ismember(A,B,'rows')	Treats each row of A and each row of B as single entities and returns a vector containing 1 (true) where the rows of matrix A are also rows of B. Elsewhere, it returns 0 (false).
issorted(A)	Returns logical 1 (true) if the elements of A are in sorted order and logical 0 (false) otherwise. Input A can be a vector or an N-by-1 or 1-by-N cell array of strings. A is considered to be sorted if A and the output of sort(A) are equal.
issorted(A, 'rows')	Returns logical 1 (true) if the rows of two-dimensional matrix A is in sorted order, and logical 0 (false) otherwise. Matrix A is considered to be sorted if A and the output of sortrows(A) are equal.
setdiff(A,B)	Sets difference of two arrays; returns the values in A that are not in B. The values in the returned array are in sorted order.
setdiff(A,B,'rows')	Treats each row of A and each row of B as single entities and returns the rows from A that are not in B. The rows of the returned matrix are in sorted order. The 'rows' option does not support cell arrays.
Setxor	Sets exclusive OR of two arrays
Union	Sets union of two arrays
unique	Unique values in array

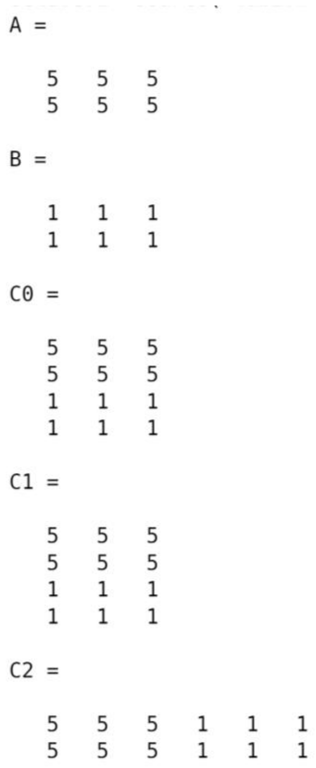
<u>CODE:</u>

(a)

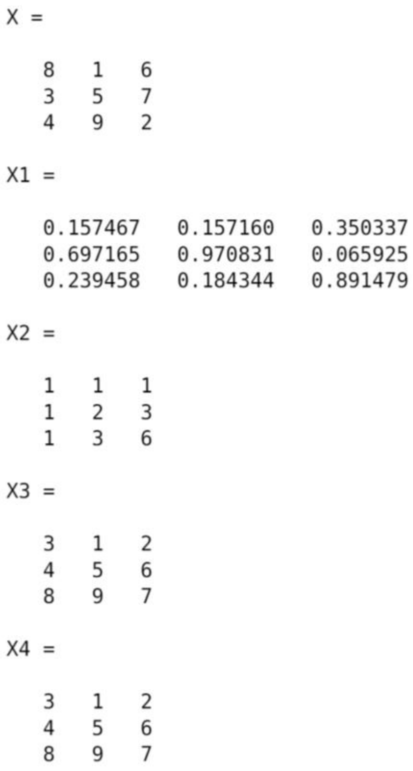
```
A=ones(2,3)*5;
display(A);
B=ones(2,3);
display(B);
C0=[A;B];
C1=cat(1,A,B);
C2=cat(2,A,B);
display(C0);
display(C1);
display(C2);
```

```
X=magic(3);
display(X);
X1=rand(3);
display(X1);
X2=pascal(3);
display(X2);
X3=sort(X);
display(X3);
X4=sort(X,'ascend');
display(X4);
X5=sort(X,'descend');
display(X5);
X6=sortrows(X,1);
display(X6);
A=[1 2 3 4 5 6 7 8 9 10 11 12];
display(A);
B=reshape(A,2,6);
display(B);
C=reshape(A,6,2);
display(C);
A1=rot90(A);
display(A1);
A2=fliplr(A);
display(A2);
```

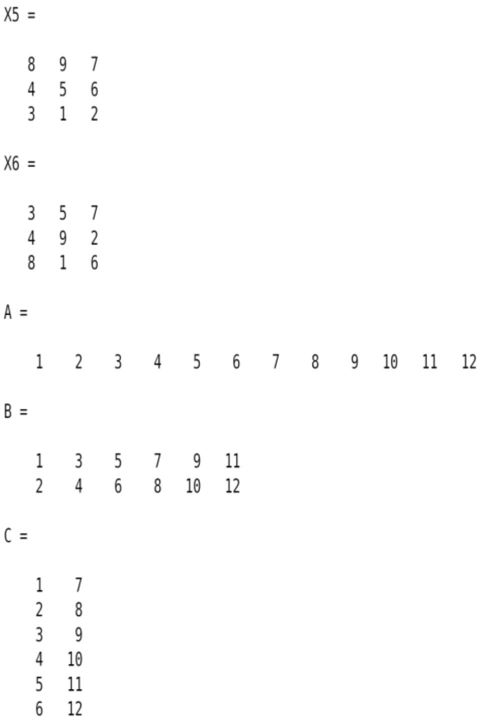
RESULT:



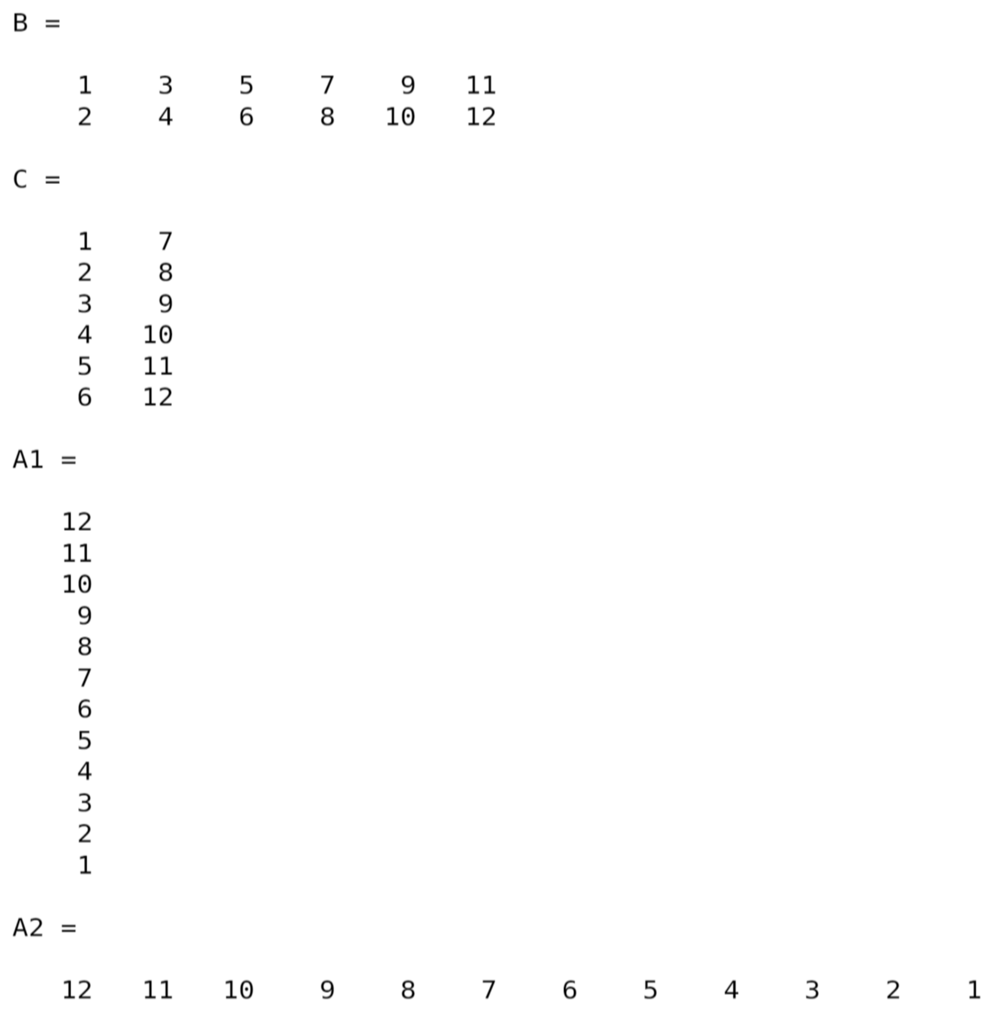
```
A =

   5   5   5
   5   5   5

B =

   1   1   1
   1   1   1

C0 =

   5   5   5
   5   5   5
   1   1   1
   1   1   1

C1 =

   5   5   5
   5   5   5
   1   1   1
   1   1   1

C2 =

   5   5   5   1   1   1
   5   5   5   1   1   1
```



```
X =

   8   1   6
   3   5   7
   4   9   2

X1 =

   0.157467   0.157160   0.350337
   0.697165   0.970831   0.065925
   0.239458   0.184344   0.891479

X2 =

   1   1   1
   1   2   3
   1   3   6

X3 =

   3   1   2
   4   5   6
   8   9   7

X4 =

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



```
X5 =

   8   9   7
   4   5   6
   3   1   2

X6 =

   3   5   7
   4   9   2
   8   1   6

A =

    1    2    3    4    5    6    7    8    9   10   11   12

B =

    1    3    5    7    9   11
    2    4    6    8   10   12

C =

    1    7
    2    8
    3    9
    4   10
    5   11
    6   12
```



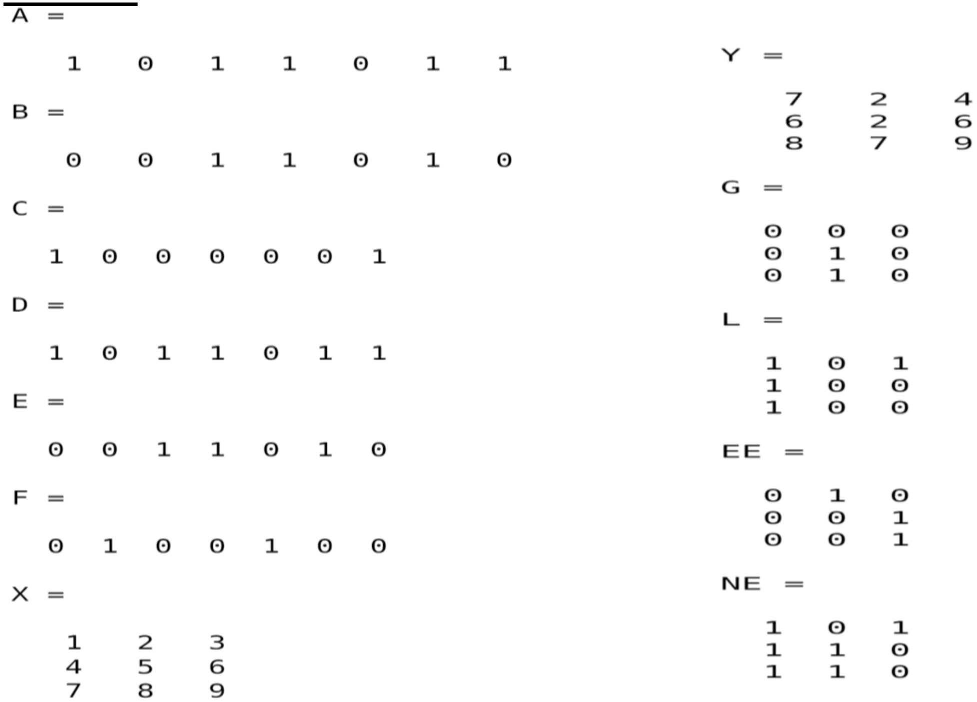
```
B =

     1     3     5     7     9    11
     2     4     6     8    10    12

C =

     1     7
     2     8
     3     9
     4    10
     5    11
     6    12

A1 =

    12
    11
    10
     9
     8
     7
     6
     5
     4
     3
     2
     1

A2 =

    12    11    10     9     8     7     6     5     4     3     2     1
```

<u>CODE:</u>

(b)

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

B=[0 0 1 1 0 1 0];

display(A);

display(B);

C=xor(A,B);

D=or(A,B);

E=and(A,B);

F=not(A);

display(C);
display(D);

display(E);

display(F);

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

Y=[7 2 4; 6 2 6; 8 7 9];

display(X);

display(Y);

G=X>Y;

L=X<Y;

EE=X==Y;

NE=X~=Y;

display(G);

display(L);

display(EE);

display(NE);
```

RESULT:



```
A =

   1   0   1   1   0   1   1

B =

   0   0   1   1   0   1   0

C =

  1  0  0  0  0  0  1

D =

  1  0  1  1  0  1  1

E =

  0  0  1  1  0  1  0

F =

  0  1  0  0  1  0  0

X =

   1   2   3
   4   5   6
   7   8   9

Y =

   7   2   4
   6   2   6
   8   7   9

G =

  0  0  0
  0  1  0
  0  1  0

L =

  1  0  1
  1  0  0
  1  0  0

EE =

  0  1  0
  0  0  1
  0  0  1

NE =

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

Criteria	Total Marks	Marks Obtained	Comments
Concept (a)	2		
Implementation (b)	2		
Performance (c)	2		
Total	6		

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.

CODE:

```
clc;
A=[1 2 3,4 5 6]
display(A,'Given matrix')
B = sum(A,1);
display(B,'Sum along column')
B = sum(A,2);
display(B,'Sum along row');
B = cumsum(A,1);
display(B,'cumsum along column')
B = cumsum(A,2;
;

display(B,'cumsum along row');
A = [4 5 6 7 8];
display(A,'Given Vector');
B = sum(A);
display(B,'SUM');
B = sum(A(2:3));
display(B,'Sum from element 2 to element 3');
B = cumsum(A);
display(B,'cumsum');
```

b)

```
clc;
clear;
close;
A=[4 5 6 7 8 1 3]
display(A);
l=length(A);
sum=0;
for i=1:1
   sum=sum+A(i);
end
display(sum);
```

c)

```
clc;
clear;
close;
A=rand(3);
display(A);
plot(A);
B=randn(3);
display(B);
plot(B);
C=randi(4,3);
display(C);
plot(C);
D=randperm(6,3);
display(D);
plot(D);
subplot(2,2,1)
plot(A);
subplot(2,2,2)
plot(B);
subplot(2,2,3)
plot(C);
subplot(2,2,4)
plot(D);
```

RESULT:

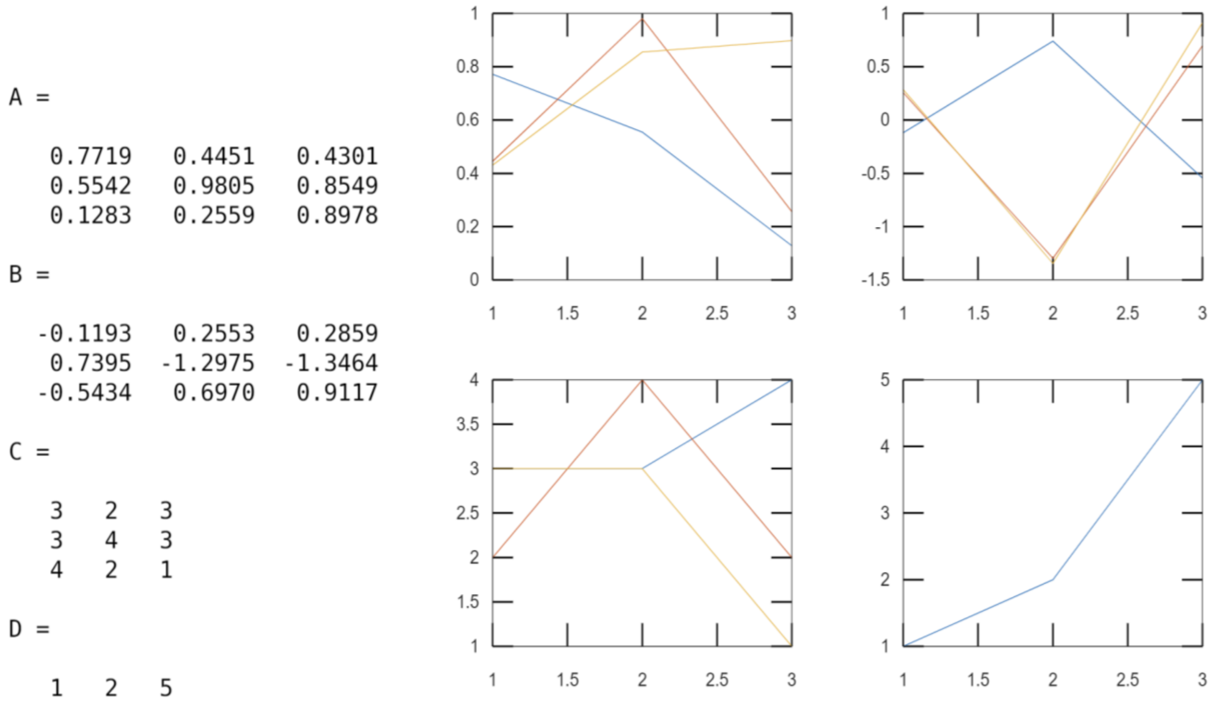
a)

```
A =

   1   2   3   4   5   6

   1   2   3   4   5   6
   1   2   3   4   5   6
21
   1   2   3   4   5   6
    1    3    6   10   15   21
   4   5   6   7   8
SUM = 30
11
cumsum =

    4    9   15   22   30
```

b)

```
A =

    4    5    6    7    8    1    3

A =

    4    5    6    7    8    1    3

sum = 4
```

c)



```
A =

   0.7719   0.4451   0.4301
   0.5542   0.9805   0.8549
   0.1283   0.2559   0.8978

B =

  -0.1193   0.2553   0.2859
   0.7395  -1.2975  -1.3464
  -0.5434   0.6970   0.9117

C =

   3   2   3
   3   4   3
   4   2   1

D =

   1   2   5
```

CONCLUSION:

The sum and the running sum is calculated successfully.

Criteria	Total Marks	Marks Obtained	Comments
Concept (a)	2		
Implementation (b)	2		
Performance (c)	2		
Total	6		

EXPERIMENT 4

AIM:

Evaluating a given expression and rounding it to the nearest integer value using Round, Floor, Ceil and Fix functions; Also, generating and Plots of (A)
Trigonometric Functions - sin(t),cos(t), tan(t), sec(t), cosec(t) and cot(t) for a given duration, ‘t’. (B)
Logarithmic and other Functions – log(A), log10(A), Square root of A, Real nth root of A.

CODE:

a)

```
clc;
clear;
close;
a=[-2.5 5.7 -1.3 4.3];
b=round(a)
c=ceil(a)
d=floor(a)
e=fix(a)
```

b) Matlab programme (using ‘hold on ’ & ‘hold off’):-

```
clc;
clear;
close;
t=-3*pi:pi/10:3*pi;
y=sin (t);
stem(t,y,'r==')
xlabel('time')
ylabel('y')
hold on
z=cos(t);
plot(t,z,'k')
a=tan(t)
plot(t,a,'m')
b=cot(t)
d=sec(t)
plot(t,d,'c--')
hold off
legend('sin','cosine','tan','cot','cosec','sec')
xlim([-5,5])
ylim([-1.5,1.5])
```

c)

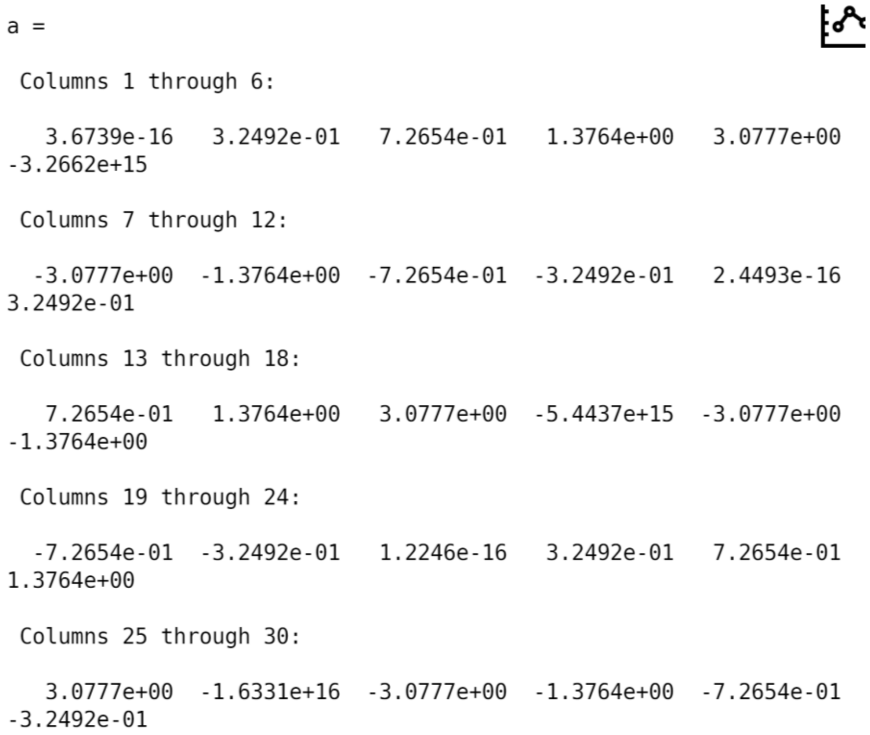
```
clc;
clear;
close;
t=-3*pi:pi/10:3*pi;
y=sin(t);
subplot(2,3,1)
stem(t,y)
title('sin')
xlabel('time')
ylabel('y')
xlim([-5,5])
ylim([-1.5,1.5])

z=cos(t);
subplot(2,3,2)
plot(t,z,'k')
title('cos')
xlim([-5,5])
ylim([-1.5,1.5])

a=tan(t);
subplot(2,3,3)
plot(t,a,'m')
title('tan')
xlim([-5,5])
ylim([-1.5,1.5])

b=cot(t);
subplot(2,3,4)
plot(t,b,'b--')
title('cot')
xlim([-5,5])
ylim([-1.5,1.5])

c=csc(t);
subplot(2,3,5)
plot(t,c,'y')
title('cosec')
xlim([-5,5])
ylim([-1.5,1.5])

d=sec(t);
subplot(2,3,6)
plot(t,d,'k--')
title('sec')
xlim([-5,5])
ylim([-1.5,1.5])
```

RESULT:

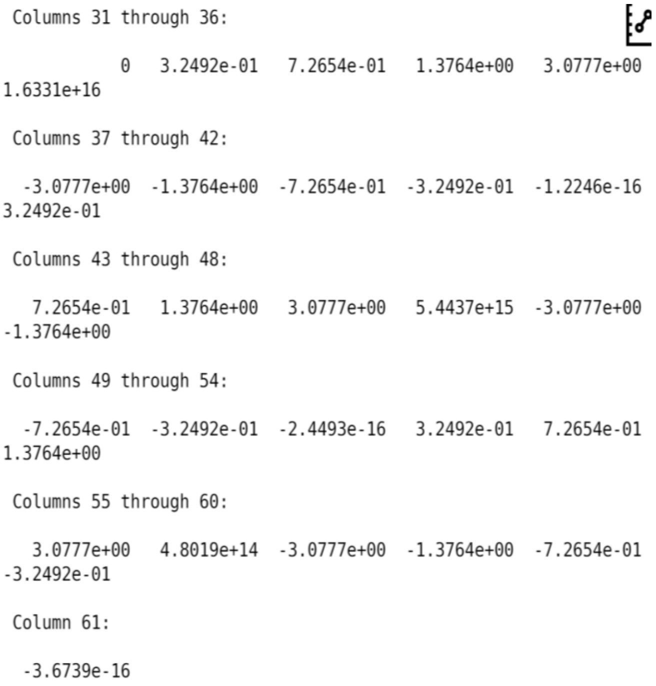
a)

```
b =

   -3    6   -1    4

c =

   -2    6   -1    5

d =

   -3    5   -2    4

e =

   -2    5   -1    4
```

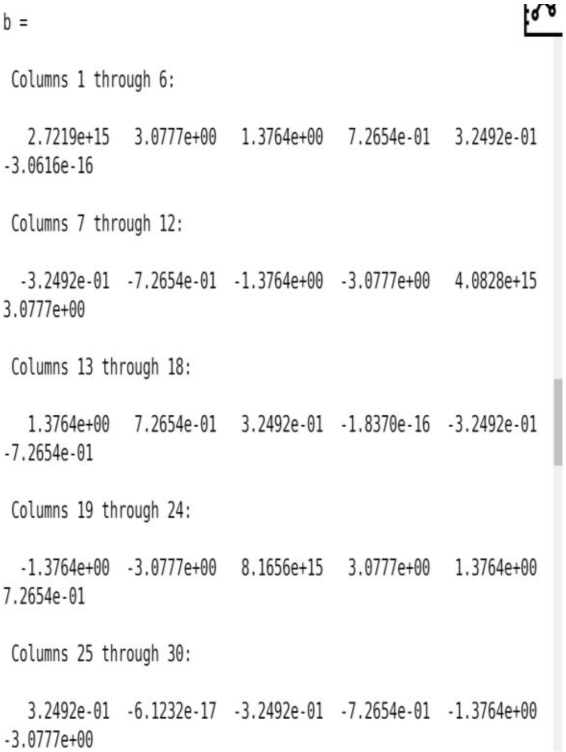
b)



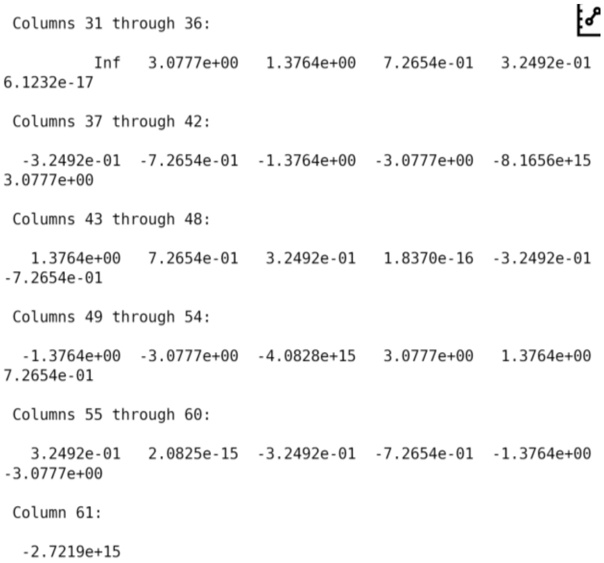
```
a =

 Columns 1 through 6:

   3.6739e-16   3.2492e-01   7.2654e-01   1.3764e+00   3.0777e+00
-3.2662e+15

 Columns 7 through 12:

  -3.0777e+00  -1.3764e+00  -7.2654e-01  -3.2492e-01   2.4493e-16
3.2492e-01

 Columns 13 through 18:

   7.2654e-01   1.3764e+00   3.0777e+00  -5.4437e+15  -3.0777e+00
-1.3764e+00

 Columns 19 through 24:

  -7.2654e-01  -3.2492e-01   1.2246e-16   3.2492e-01   7.2654e-01
1.3764e+00

 Columns 25 through 30:

   3.0777e+00  -1.6331e+16  -3.0777e+00  -1.3764e+00  -7.2654e-01
-3.2492e-01
```



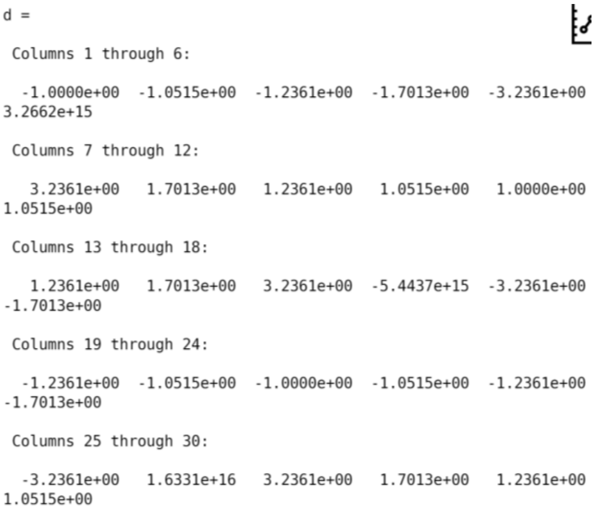
```
 Columns 31 through 36:

           0   3.2492e-01   7.2654e-01   1.3764e+00   3.0777e+00
1.6331e+16

 Columns 37 through 42:

  -3.0777e+00  -1.3764e+00  -7.2654e-01  -3.2492e-01  -1.2246e-16
3.2492e-01

 Columns 43 through 48:

   7.2654e-01   1.3764e+00   3.0777e+00   5.4437e+15  -3.0777e+00
-1.3764e+00

 Columns 49 through 54:

  -7.2654e-01  -3.2492e-01  -2.4493e-16   3.2492e-01   7.2654e-01
1.3764e+00

 Columns 55 through 60:

   3.0777e+00   4.8019e+14  -3.0777e+00  -1.3764e+00  -7.2654e-01
-3.2492e-01

 Column 61:

  -3.6739e-16
```



```
b =

 Columns 1 through 6:

   2.7219e+15   3.0777e+00   1.3764e+00   7.2654e-01   3.2492e-01
-3.0616e-16

 Columns 7 through 12:

  -3.2492e-01  -7.2654e-01  -1.3764e+00  -3.0777e+00   4.0828e+15
3.0777e+00

 Columns 13 through 18:

   1.3764e+00   7.2654e-01   3.2492e-01  -1.8370e-16  -3.2492e-01
-7.2654e-01

 Columns 19 through 24:

  -1.3764e+00  -3.0777e+00   8.1656e+15   3.0777e+00   1.3764e+00
7.2654e-01

 Columns 25 through 30:

   3.2492e-01  -6.1232e-17  -3.2492e-01  -7.2654e-01  -1.3764e+00
-3.0777e+00
```



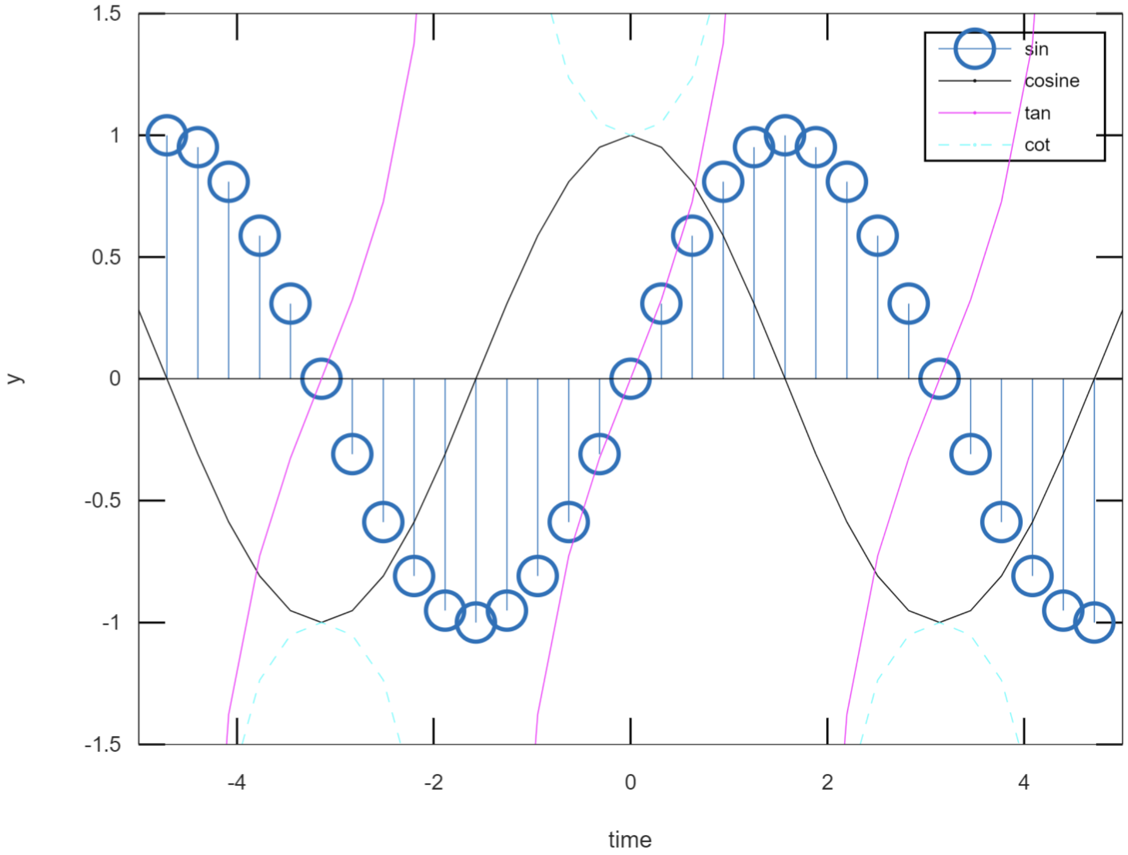
```
 Columns 31 through 36:

         Inf   3.0777e+00   1.3764e+00   7.2654e-01   3.2492e-01
6.1232e-17

 Columns 37 through 42:

  -3.2492e-01  -7.2654e-01  -1.3764e+00  -3.0777e+00  -8.1656e+15
3.0777e+00

 Columns 43 through 48:

   1.3764e+00   7.2654e-01   3.2492e-01   1.8370e-16  -3.2492e-01
-7.2654e-01

 Columns 49 through 54:

  -1.3764e+00  -3.0777e+00  -4.0828e+15   3.0777e+00   1.3764e+00
7.2654e-01

 Columns 55 through 60:

   3.2492e-01   2.0825e-15  -3.2492e-01  -7.2654e-01  -1.3764e+00
-3.0777e+00

 Column 61:

  -2.7219e+15
```



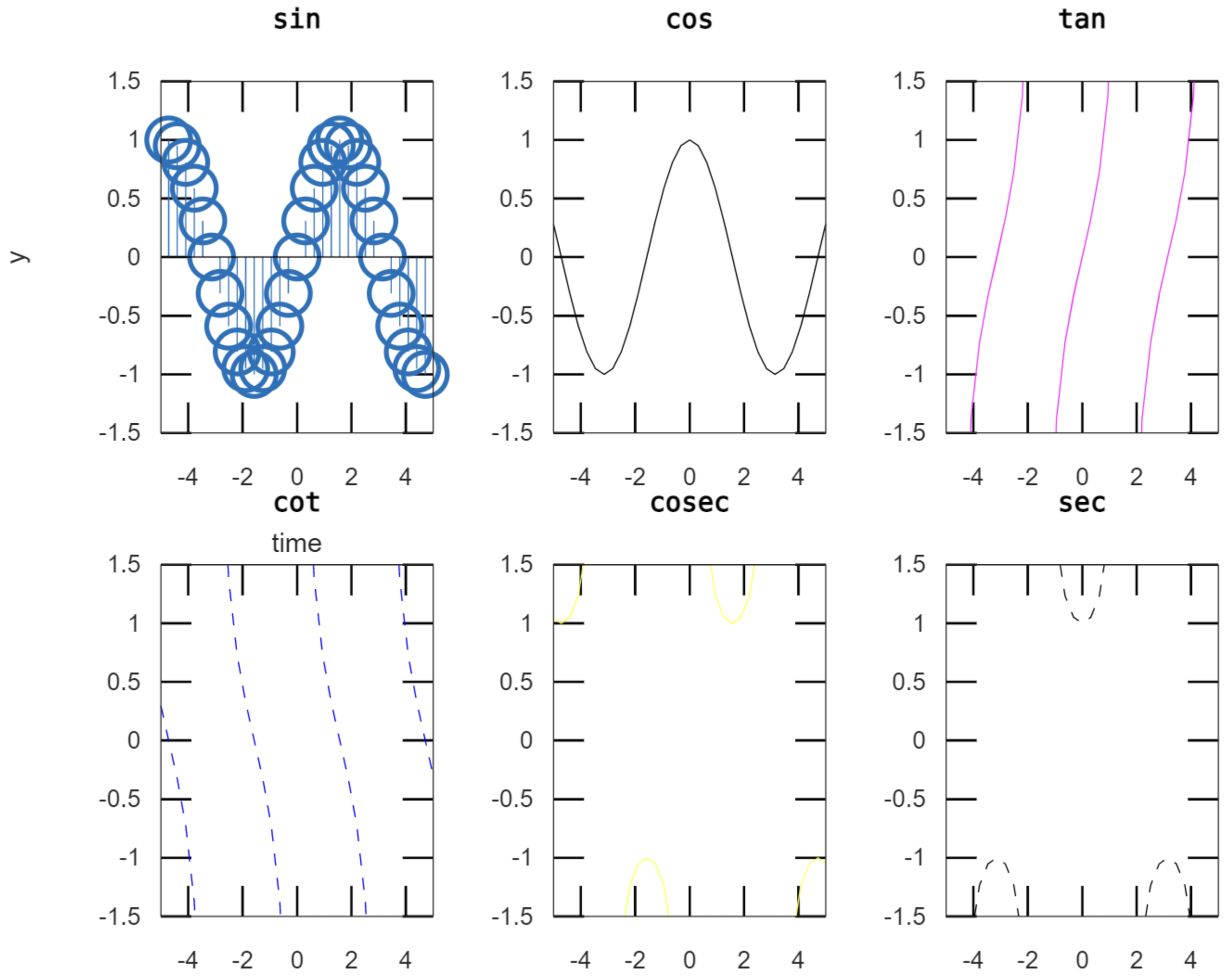
```
d =

 Columns 1 through 6:

  -1.0000e+00  -1.0515e+00  -1.2361e+00  -1.7013e+00  -3.2361e+00
3.2662e+15

 Columns 7 through 12:

   3.2361e+00   1.7013e+00   1.2361e+00   1.0515e+00   1.0000e+00
1.0515e+00

 Columns 13 through 18:

   1.2361e+00   1.7013e+00   3.2361e+00  -5.4437e+15  -3.2361e+00
-1.7013e+00

 Columns 19 through 24:

  -1.2361e+00  -1.0515e+00  -1.0000e+00  -1.0515e+00  -1.2361e+00
-1.7013e+00

 Columns 25 through 30:

  -3.2361e+00   1.6331e+16   3.2361e+00   1.7013e+00   1.2361e+00
1.0515e+00
```

```
 Columns 31 through 36:

   1.0000e+00   1.0515e+00   1.2361e+00   1.7013e+00   3.2361e+00
1.6331e+16

 Columns 37 through 42:

  -3.2361e+00  -1.7013e+00  -1.2361e+00  -1.0515e+00  -1.0000e+00
-1.0515e+00

 Columns 43 through 48:

  -1.2361e+00  -1.7013e+00  -3.2361e+00  -5.4437e+15   3.2361e+00
1.7013e+00

 Columns 49 through 54:

   1.2361e+00   1.0515e+00   1.0000e+00   1.0515e+00   1.2361e+00
1.7013e+00

 Columns 55 through 60:

   3.2361e+00   4.8019e+14  -3.2361e+00  -1.7013e+00  -1.2361e+00
-1.0515e+00

 Column 61:

  -1.0000e+00
```

```
ans = -13.513
```



c)



CONCLUSION:

The graphs has been plotted successfully.

Criteria	Total Marks	Marks Obtained	Comments
Concept (a)	2		
Implementation (b)	2		
Performance (c)	2		
Total	6		

EXPERIMENT 5

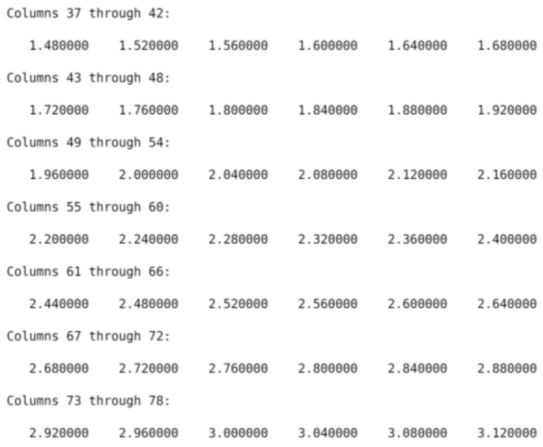
AIM:

Creating a vector X with elements, Xn = (-1)n+1/(2n-1) and Adding up 100 elements of the vector, X; And, plotting the functions, x, x3, ex, exp(x2) over the interval 0 < x < 4 (by choosing appropriate mesh values for x to obtain smooth curves), on A Rectangular Plot

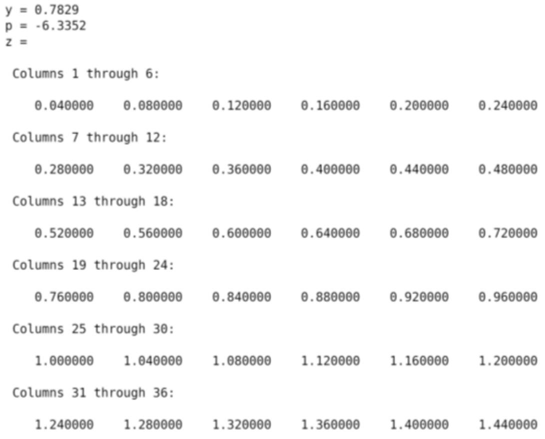
CODE:

```
for n=1:100
   x(n)=((-1)^(n+1))/(2*n-1);
end
y=sum(x)
p=plot(x)
z=0.04:0.04:4
plot(z,x)

xlabel('values of z')
ylabel('x(n)')
title('Plot of x(n)')
p=x.^3
q=exp(x)
r=exp(x.^2)
```

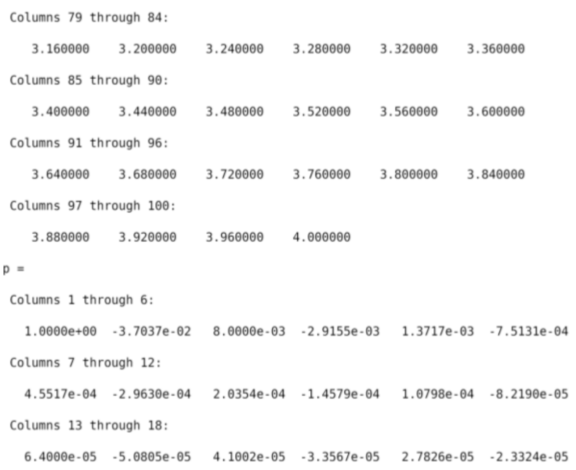
OBSERVATIONS:



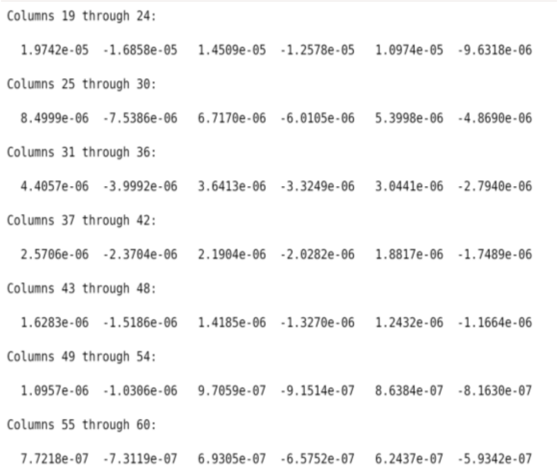
```
Columns 37 through 42:

   1.480000    1.520000    1.560000    1.600000    1.640000    1.680000

Columns 43 through 48:

   1.720000    1.760000    1.800000    1.840000    1.880000    1.920000

Columns 49 through 54:

   1.960000    2.000000    2.040000    2.080000    2.120000    2.160000

Columns 55 through 60:

   2.200000    2.240000    2.280000    2.320000    2.360000    2.400000

Columns 61 through 66:

   2.440000    2.480000    2.520000    2.560000    2.600000    2.640000

Columns 67 through 72:

   2.680000    2.720000    2.760000    2.800000    2.840000    2.880000

Columns 73 through 78:

   2.920000    2.960000    3.000000    3.040000    3.080000    3.120000
```



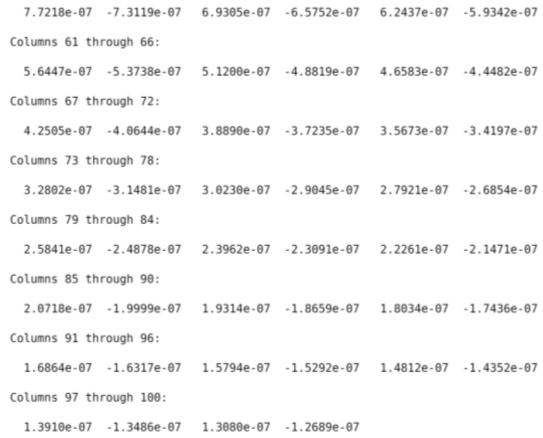
```
y = 0.7829
p = -6.3352
z =

 Columns 1 through 6:

   0.040000    0.080000    0.120000    0.160000    0.200000    0.240000

 Columns 7 through 12:

   0.280000    0.320000    0.360000    0.400000    0.440000    0.480000

 Columns 13 through 18:

   0.520000    0.560000    0.600000    0.640000    0.680000    0.720000

 Columns 19 through 24:

   0.760000    0.800000    0.840000    0.880000    0.920000    0.960000

 Columns 25 through 30:

   1.000000    1.040000    1.080000    1.120000    1.160000    1.200000

 Columns 31 through 36:

   1.240000    1.280000    1.320000    1.360000    1.400000    1.440000
```



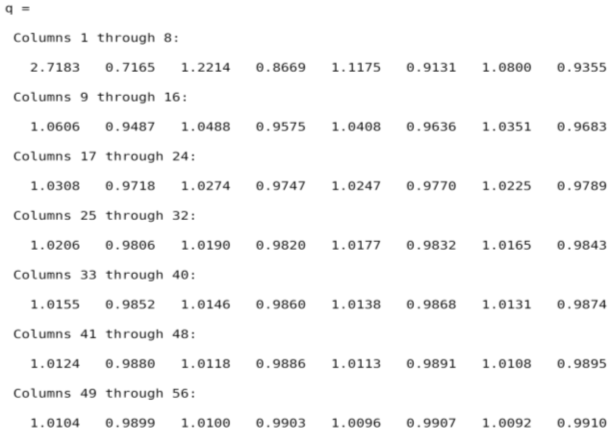
```
 Columns 79 through 84:

    3.160000    3.200000    3.240000    3.280000    3.320000    3.360000

 Columns 85 through 90:

    3.400000    3.440000    3.480000    3.520000    3.560000    3.600000

 Columns 91 through 96:

    3.640000    3.680000    3.720000    3.760000    3.800000    3.840000

 Columns 97 through 100:

    3.880000    3.920000    3.960000    4.000000

p =

 Columns 1 through 6:

   1.0000e+00  -3.7037e-02   8.0000e-03  -2.9155e-03   1.3717e-03  -7.5131e-04

 Columns 7 through 12:

   4.5517e-04  -2.9630e-04   2.0354e-04  -1.4579e-04   1.0798e-04  -8.2190e-05

 Columns 13 through 18:

   6.4000e-05  -5.0805e-05   4.1002e-05  -3.3567e-05   2.7826e-05  -2.3324e-05
```



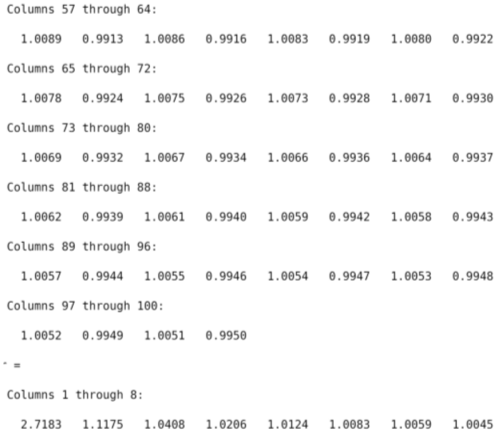
```
 Columns 19 through 24:

   1.9742e-05  -1.6858e-05   1.4509e-05  -1.2578e-05   1.0974e-05  -9.6318e-06

 Columns 25 through 30:

   8.4999e-06  -7.5386e-06   6.7170e-06  -6.0105e-06   5.3998e-06  -4.8690e-06

 Columns 31 through 36:

   4.4057e-06  -3.9992e-06   3.6413e-06  -3.3249e-06   3.0441e-06  -2.7940e-06

 Columns 37 through 42:

   2.5706e-06  -2.3704e-06   2.1904e-06  -2.0282e-06   1.8817e-06  -1.7489e-06

 Columns 43 through 48:

   1.6283e-06  -1.5186e-06   1.4185e-06  -1.3270e-06   1.2432e-06  -1.1664e-06

 Columns 49 through 54:

   1.0957e-06  -1.0306e-06   9.7059e-07  -9.1514e-07   8.6384e-07  -8.1630e-07

 Columns 55 through 60:

   7.7218e-07  -7.3119e-07   6.9305e-07  -6.5752e-07   6.2437e-07  -5.9342e-07
```



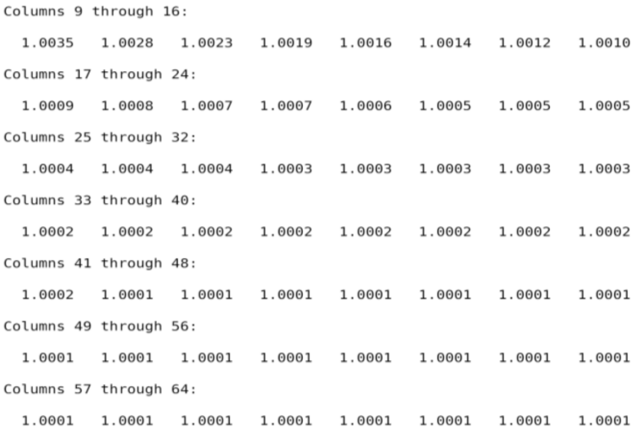
```
   7.7218e-07  -7.3119e-07   6.9305e-07  -6.5752e-07   6.2437e-07  -5.9342e-07

 Columns 61 through 66:

   5.6447e-07  -5.3738e-07   5.1200e-07  -4.8819e-07   4.6583e-07  -4.4482e-07

 Columns 67 through 72:

   4.2505e-07  -4.0644e-07   3.8890e-07  -3.7235e-07   3.5673e-07  -3.4197e-07

 Columns 73 through 78:

   3.2802e-07  -3.1481e-07   3.0230e-07  -2.9045e-07   2.7921e-07  -2.6854e-07

 Columns 79 through 84:

   2.5841e-07  -2.4878e-07   2.3962e-07  -2.3091e-07   2.2261e-07  -2.1471e-07

 Columns 85 through 90:

   2.0718e-07  -1.9999e-07   1.9314e-07  -1.8659e-07   1.8034e-07  -1.7436e-07

 Columns 91 through 96:

   1.6864e-07  -1.6317e-07   1.5794e-07  -1.5292e-07   1.4812e-07  -1.4352e-07

 Columns 97 through 100:

   1.3910e-07  -1.3486e-07   1.3080e-07  -1.2689e-07
```



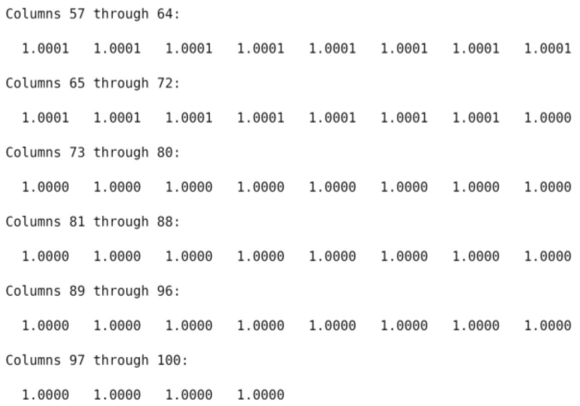
```
q =

 Columns 1 through 8:

   2.7183   0.7165   1.2214   0.8669   1.1175   0.9131   1.0800   0.9355

 Columns 9 through 16:

   1.0606   0.9487   1.0488   0.9575   1.0408   0.9636   1.0351   0.9683

 Columns 17 through 24:

   1.0308   0.9718   1.0274   0.9747   1.0247   0.9770   1.0225   0.9789

 Columns 25 through 32:

   1.0206   0.9806   1.0190   0.9820   1.0177   0.9832   1.0165   0.9843

 Columns 33 through 40:

   1.0155   0.9852   1.0146   0.9860   1.0138   0.9868   1.0131   0.9874

 Columns 41 through 48:

   1.0124   0.9880   1.0118   0.9886   1.0113   0.9891   1.0108   0.9895

 Columns 49 through 56:

   1.0104   0.9899   1.0100   0.9903   1.0096   0.9907   1.0092   0.9910
```



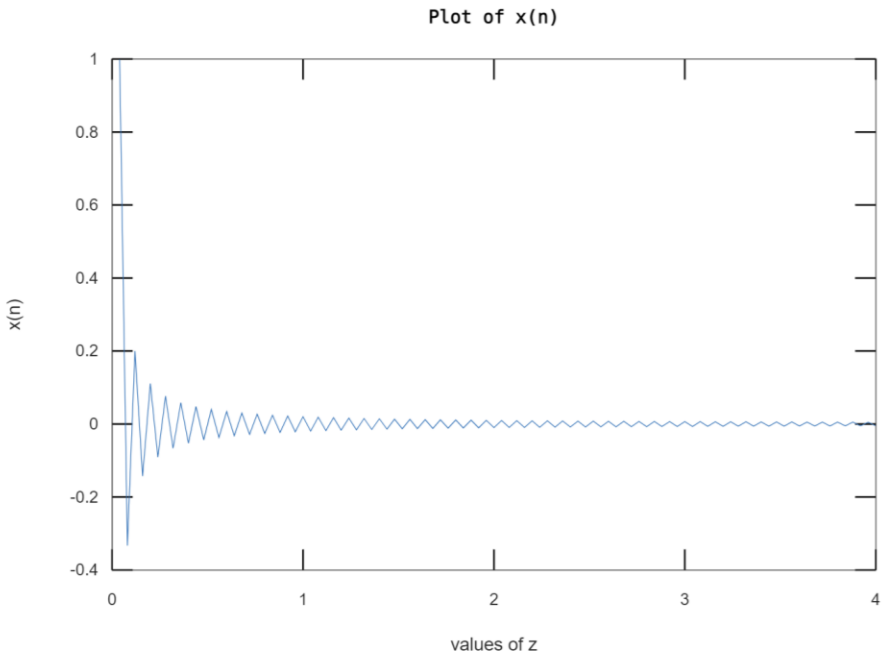
```
 Columns 57 through 64:

   1.0089   0.9913   1.0086   0.9916   1.0083   0.9919   1.0080   0.9922

 Columns 65 through 72:

   1.0078   0.9924   1.0075   0.9926   1.0073   0.9928   1.0071   0.9930

 Columns 73 through 80:

   1.0069   0.9932   1.0067   0.9934   1.0066   0.9936   1.0064   0.9937

 Columns 81 through 88:

   1.0062   0.9939   1.0061   0.9940   1.0059   0.9942   1.0058   0.9943

 Columns 89 through 96:

   1.0057   0.9944   1.0055   0.9946   1.0054   0.9947   1.0053   0.9948

 Columns 97 through 100:

   1.0052   0.9949   1.0051   0.9950

̂ =

 Columns 1 through 8:

   2.7183   1.1175   1.0408   1.0206   1.0124   1.0083   1.0059   1.0045
```



```
Columns 9 through 16:

  1.0035   1.0028   1.0023   1.0019   1.0016   1.0014   1.0012   1.0010

Columns 17 through 24:

  1.0009   1.0008   1.0007   1.0007   1.0006   1.0005   1.0005   1.0005

Columns 25 through 32:

  1.0004   1.0004   1.0004   1.0003   1.0003   1.0003   1.0003   1.0003

Columns 33 through 40:

  1.0002   1.0002   1.0002   1.0002   1.0002   1.0002   1.0002   1.0002

Columns 41 through 48:

  1.0002   1.0001   1.0001   1.0001   1.0001   1.0001   1.0001   1.0001

Columns 49 through 56:

  1.0001   1.0001   1.0001   1.0001   1.0001   1.0001   1.0001   1.0001

Columns 57 through 64:

  1.0001   1.0001   1.0001   1.0001   1.0001   1.0001   1.0001   1.0001
```



```
Columns 57 through 64:

  1.0001   1.0001   1.0001   1.0001   1.0001   1.0001   1.0001   1.0001

Columns 65 through 72:

  1.0001   1.0001   1.0001   1.0001   1.0001   1.0001   1.0001   1.0000

Columns 73 through 80:

  1.0000   1.0000   1.0000   1.0000   1.0000   1.0000   1.0000   1.0000

Columns 81 through 88:

  1.0000   1.0000   1.0000   1.0000   1.0000   1.0000   1.0000   1.0000

Columns 89 through 96:

  1.0000   1.0000   1.0000   1.0000   1.0000   1.0000   1.0000   1.0000

Columns 97 through 100:

  1.0000   1.0000   1.0000   1.0000
```



DISCUSSION:

The vector was constructed using for loop and the given formula. And the sum of all the elements was calculated using sum().Then the functions x,x^3 ,e^x ,exp(x^2) were plotted over the interval 0<x<4 using plot().

RESULT:

Vector x was created and subsequently x, x^3 , e^x ,exp(x^2) were plotted over the interval

0<x<4.

CONCLUSION:

Several functions x, x^3 , e^x ,exp(x^2) were successfully plotted for a vector x.

Criteria	Total Marks	Marks Obtained	Comments
Concept (a)	2		
Implementation (b)	2		
Performance (c)	2		
Total	6		

EXPERIMENT 6

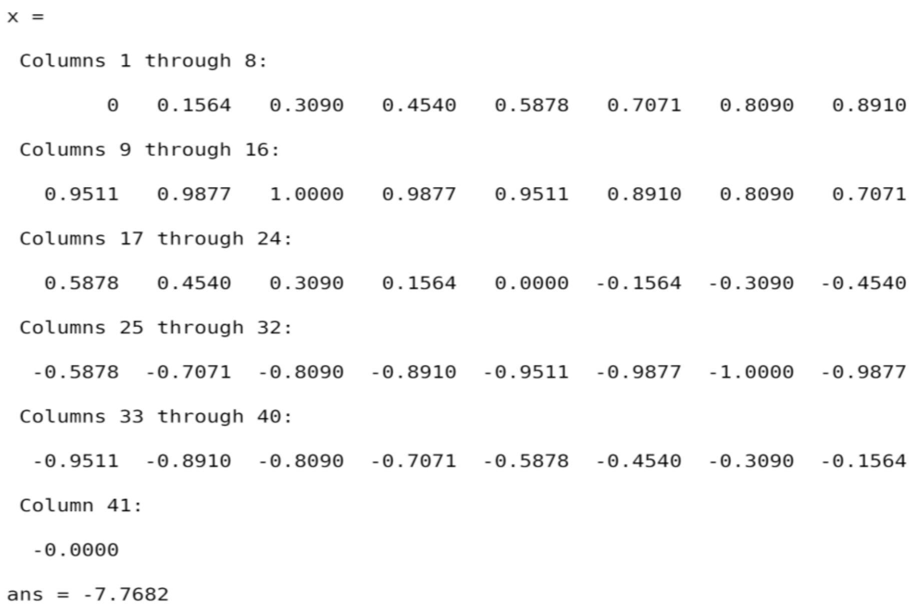
AIM:

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

CODE:

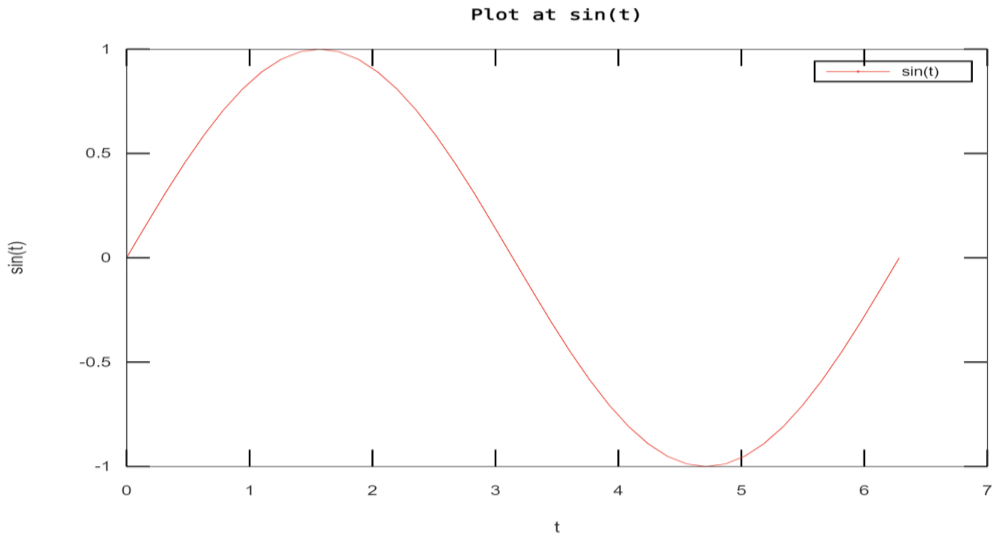
```
t=0:0.05*pi:2*pi;
x=sin(t)
plot(t,x,'r-')
xlabel('t')
ylabel('sin(t)')
title('Plot at sin(t)')
legend('sin(t)')
text(pi,0,'fontsize',18)
hold on
y=sin(3*t)
plot(t,y,'b+-')
xlabel('Radians')
ylabel('Amplitude')
title('plot of sin (3{/alpha})over existing curve sin({/alpha})')
legend('sin(t)','sin(3t)')
t=0:0.05*pi:2*pi,
text(4.1,-0.8,'fontsize',18)
text(1,0.2,'fontsize',18)
```

OBSERVATION:



```
x =

 Columns 1 through 8:

       0   0.1564   0.3090   0.4540   0.5878   0.7071   0.8090   0.8910

 Columns 9 through 16:

   0.9511   0.9877   1.0000   0.9877   0.9511   0.8910   0.8090   0.7071

 Columns 17 through 24:

   0.5878   0.4540   0.3090   0.1564   0.0000  -0.1564  -0.3090  -0.4540

 Columns 25 through 32:

  -0.5878  -0.7071  -0.8090  -0.8910  -0.9511  -0.9877  -1.0000  -0.9877

 Columns 33 through 40:

  -0.9511  -0.8910  -0.8090  -0.7071  -0.5878  -0.4540  -0.3090  -0.1564

 Column 41:

  -0.0000

ans = -7.7682
```

PLOT:



RESULT:

The given Experiment is performed successfully.

Criteria	Total Marks	Marks Obtained	Comments
Concept (a)	2		
Implementation (b)	2		
Performance (c)	2		
Total	6		

EXPERIMENT 7

AIM:

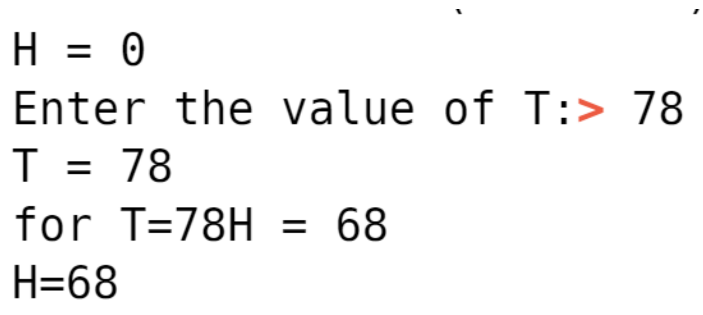
Writing brief Scripts starting each Script with a request for input (using input) to Evaluate the function h(T) using if-else statement, where,
h(T) = (T – 10) for 0 < T < 100
h(T) = (0.45 T + 900) for T > 100.

CODE:

```
H=0
T=input('Enter the value of T:' )
if(T<0)
   disp('Enter a value of T greater than 0')
else if(0<T&&T<100)
   fprintf('for T=%d',T)
   H=T-10
else if(T>100)
   fprintf('for T=%d',T)
H=((0.45*T)+900)
end
end
end
fprintf('H=%d',H)
```

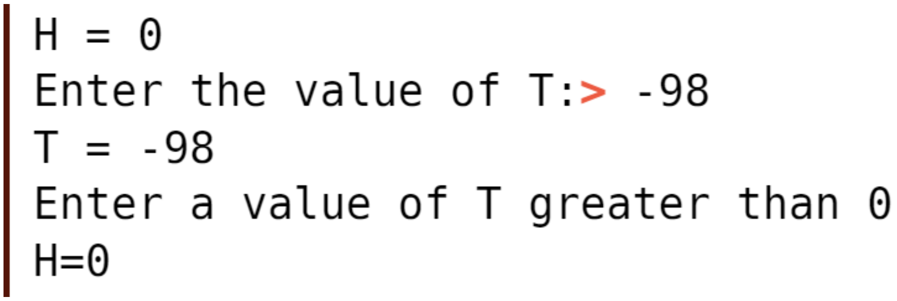
RESULT:

FOR T>0:



```
H = 0
Enter the value of T:> 78
T = 78
for T=78H = 68
H=68
```

FOR T<0:



```
H = 0
Enter the value of T:> -98
T = -98
Enter a value of T greater than 0
H=0
```

CONCLUSION:

The experiment calculated value of T and H according to the given if-else condition.

Criteria	Total Marks	Marks Obtained	Comments
Concept (a)	2		
Implementation (b)	2		
Performance (c)	2		
Total	6		

EXPERIMENT 8

AIM:

Solving First, Second and third Order Ordinary Differential Equation using Built-in Functions and plot.
Consider the following ordinary differential equation:

$$x\left(\frac{dy}{dx}\right) + 2y = x^3$$

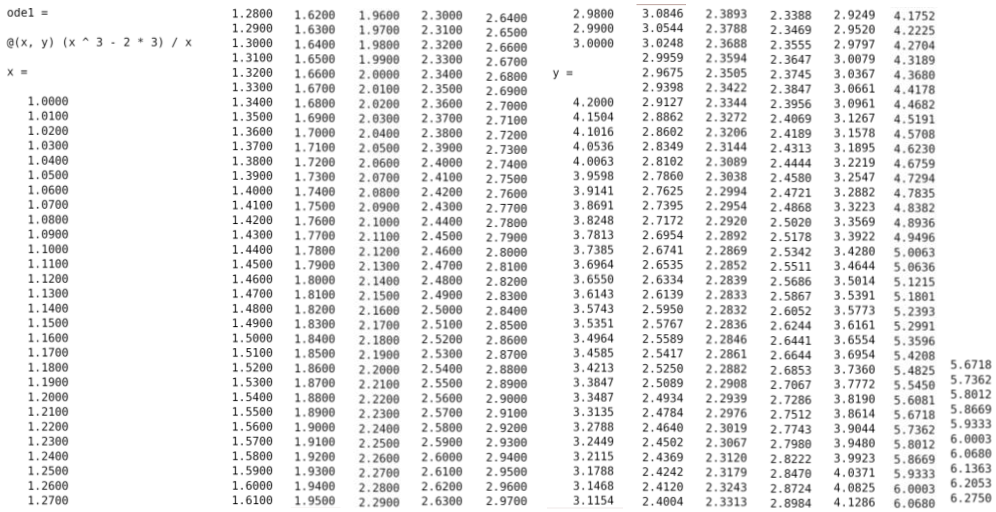
CODE:

```
ode1=@(x,y)(x^3-2*3)/x
[x,y]=ode45(ode1,[1:0.01:3],4.2)
plot(x,y,'linewidth',2)
xlabel('x')
ylabel('y')
grid on
title('solution to ODE,dy/dx=(x^3-2*3)/x')
```

DISCUSSION:

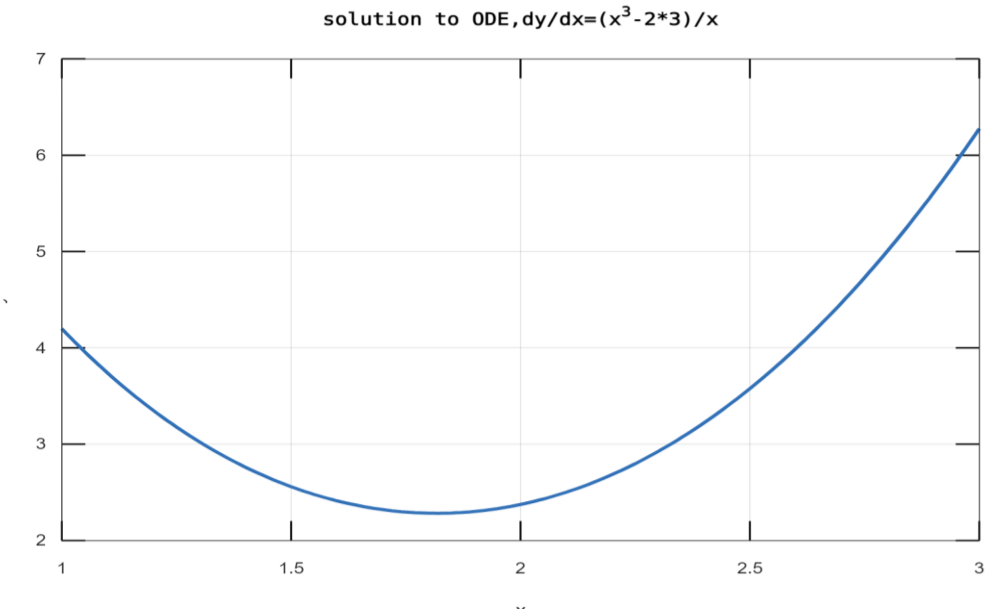
Ode45:
Ode45 is based on an explicit Runge-kutta (4,5) formula, the Dormand-Prince pair . It needs only the solution of the immediately preceding time point y(tn-1).

OBSERVATIONS:



```
ode1 =                          1.2800    1.6200    1.9600    2.3000    2.6400      2.9800    3.0846    2.3893    2.3388    2.9249    4.1752
                                1.2900    1.6300    1.9700    2.3100    2.6500      2.9900    3.0544    2.3788    2.3469    2.9520    4.2225
@(x, y) (x ^ 3 - 2 * 3) / x     1.3000    1.6400    1.9800    2.3200    2.6600      3.0000    3.0248    2.3688    2.3555    2.9797    4.2704
                                1.3100    1.6500    1.9900    2.3300    2.6700                2.9959    2.3594    2.3647    3.0079    4.3189
x =                             1.3200    1.6600    2.0000    2.3400    2.6800    y =         2.9675    2.3505    2.3745    3.0367    4.3680
                                1.3300    1.6700    2.0100    2.3500    2.6900                2.9398    2.3422    2.3847    3.0661    4.4178
    1.0000                      1.3400    1.6800    2.0200    2.3600    2.7000      4.2000    2.9127    2.3344    2.3956    3.0961    4.4682
    1.0100                      1.3500    1.6900    2.0300    2.3700    2.7100      4.1504    2.8862    2.3272    2.4069    3.1267    4.5191
    1.0200                      1.3600    1.7000    2.0400    2.3800    2.7200      4.1016    2.8602    2.3206    2.4189    3.1578    4.5708
    1.0300                      1.3700    1.7100    2.0500    2.3900    2.7300      4.0536    2.8349    2.3144    2.4313    3.1895    4.6230
    1.0400                      1.3800    1.7200    2.0600    2.4000    2.7400      4.0063    2.8102    2.3089    2.4444    3.2219    4.6759
    1.0500                      1.3900    1.7300    2.0700    2.4100    2.7500      3.9598    2.7860    2.3038    2.4580    3.2547    4.7294
    1.0600                      1.4000    1.7400    2.0800    2.4200    2.7600      3.9141    2.7625    2.2994    2.4721    3.2882    4.7835
    1.0700                      1.4100    1.7500    2.0900    2.4300    2.7700      3.8691    2.7395    2.2954    2.4868    3.3223    4.8382
    1.0800                      1.4200    1.7600    2.1000    2.4400    2.7800      3.8248    2.7172    2.2920    2.5020    3.3569    4.8936
    1.0900                      1.4300    1.7700    2.1100    2.4500    2.7900      3.7813    2.6954    2.2892    2.5178    3.3922    4.9496
    1.1000                      1.4400    1.7800    2.1200    2.4600    2.8000      3.7385    2.6741    2.2869    2.5342    3.4280    5.0063
    1.1100                      1.4500    1.7900    2.1300    2.4700    2.8100      3.6964    2.6535    2.2852    2.5511    3.4644    5.0636
    1.1200                      1.4600    1.8000    2.1400    2.4800    2.8200      3.6550    2.6334    2.2839    2.5686    3.5014    5.1215
    1.1300                      1.4700    1.8100    2.1500    2.4900    2.8300      3.6143    2.6139    2.2833    2.5867    3.5391    5.1801
    1.1400                      1.4800    1.8200    2.1600    2.5000    2.8400      3.5743    2.5950    2.2832    2.6052    3.5773    5.2393
    1.1500                      1.4900    1.8300    2.1700    2.5100    2.8500      3.5351    2.5767    2.2836    2.6244    3.6161    5.2991
    1.1600                      1.5000    1.8400    2.1800    2.5200    2.8600      3.4964    2.5589    2.2846    2.6441    3.6554    5.3596
    1.1700                      1.5100    1.8500    2.1900    2.5300    2.8700      3.4585    2.5417    2.2861    2.6644    3.6954    5.4208
    1.1800                      1.5200    1.8600    2.2000    2.5400    2.8800      3.4213    2.5250    2.2882    2.6853    3.7360    5.4825    5.6718
    1.1900                      1.5300    1.8700    2.2100    2.5500    2.8900      3.3847    2.5089    2.2908    2.7067    3.7772    5.5450    5.7362
    1.2000                      1.5400    1.8800    2.2200    2.5600    2.9000      3.3487    2.4934    2.2939    2.7286    3.8190    5.6081    5.8012
    1.2100                      1.5500    1.8900    2.2300    2.5700    2.9100      3.3135    2.4784    2.2976    2.7512    3.8614    5.6718    5.8669
    1.2200                      1.5600    1.9000    2.2400    2.5800    2.9200      3.2788    2.4640    2.3019    2.7743    3.9044    5.7362    5.9333
    1.2300                      1.5700    1.9100    2.2500    2.5900    2.9300      3.2449    2.4502    2.3067    2.7980    3.9480    5.8012    6.0003
    1.2400                      1.5800    1.9200    2.2600    2.6000    2.9400      3.2115    2.4369    2.3120    2.8222    3.9923    5.8669    6.0680
    1.2500                      1.5900    1.9300    2.2700    2.6100    2.9500      3.1788    2.4242    2.3179    2.8470    4.0371    5.9333    6.1363
    1.2600                      1.6000    1.9400    2.2800    2.6200    2.9600      3.1468    2.4120    2.3243    2.8724    4.0825    6.0003    6.2053
    1.2700                      1.6100    1.9500    2.2900    2.6300    2.9700      3.1154    2.4004    2.3313    2.8984    4.1286    6.0680    6.2750
```

RESULT:



CONCLUSION:

The solution to the ordinary differential equation using ode45 is calculated and plot is made on graph.

Criteria	Total Marks	Marks Obtained	Comments
Concept (a)	2		
Implementation (b)	2		
Performance (c)	2		
Total	6		

EXPERIMENT 9

AIM:

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

DISCUSSION:

```
*for loop
      Syntax
      Sum=0
       For i=1:5
       Sum=sum+I;
End
```

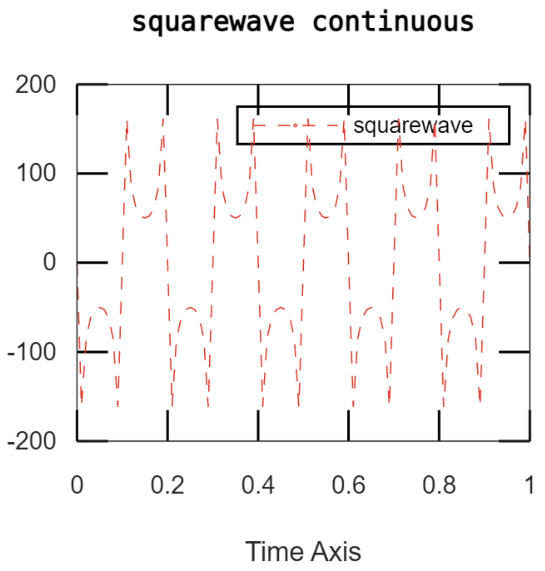
*sin function
Y=sin(x);
X in radians

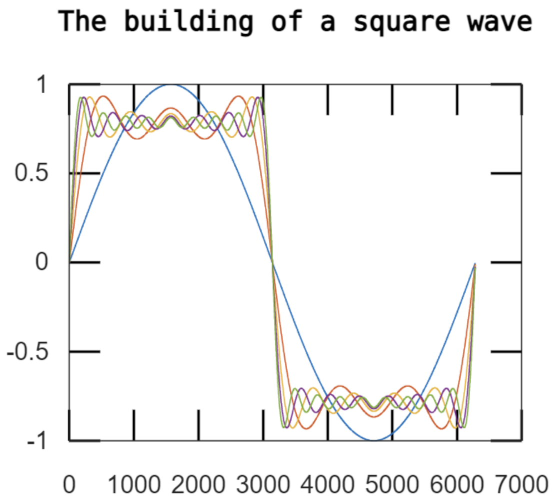
*Square wave is a waveform in which the amplitude alternates at a steady frequency
Between fixed minimum and maximum values.()

CODE:

```
t = 0:.001:6.28;
y = zeros(10,length(t));
x = zeros(size(t));
for k = 1:2:19
   x = x + sin(k*t)/k;
   y((k+1)/2,:) = x;
end
plot(y(1:2:9,:)')
title('The building of a square wave')
```

RESULT:





CONCLUSION:

The squarewave form is successfully constructed.

Criteria	Total Marks	Marks Obtained	Comments
Concept (a)	2		
Implementation (b)	2		
Performance (c)	2		
Total	6		

EXPERIMENT 10

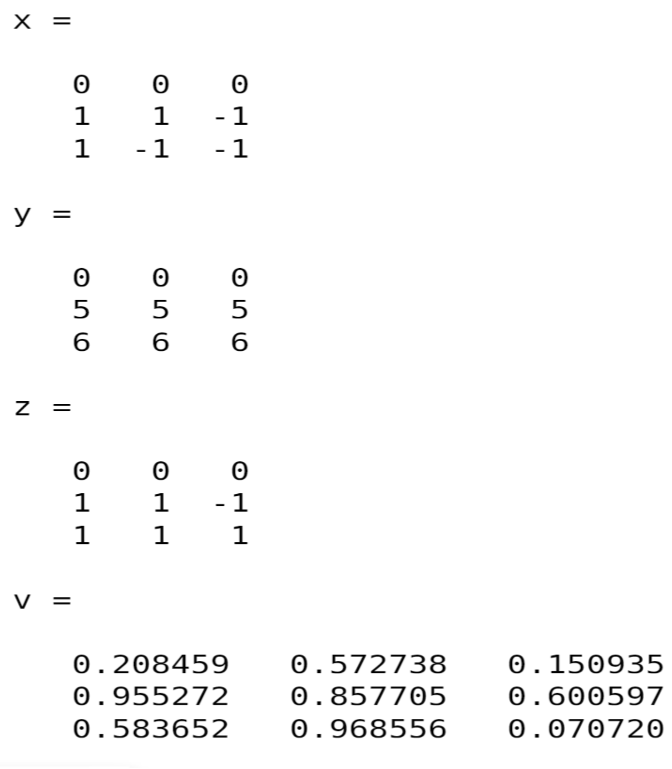
AIM:

Basic 2d and 3d plots: parametric space curve, polygons and vertices 3d contour lines, pie and bar chart.

CODE:

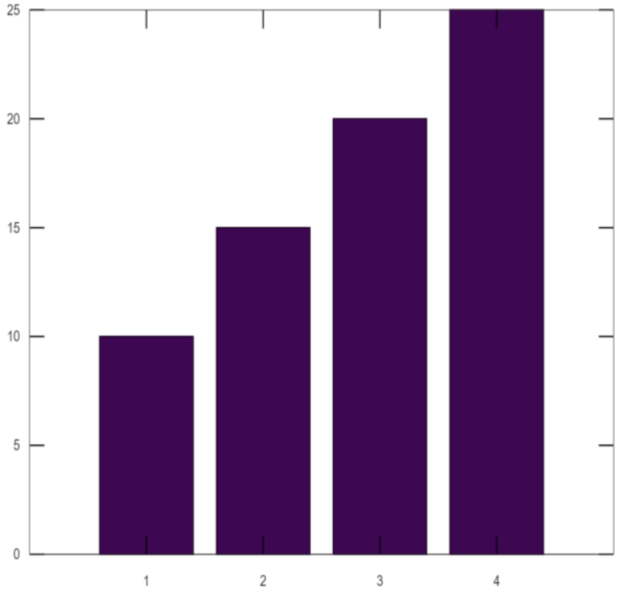
```
x = [10 15 20 25];
bar(x);
pie(x);

 x = [0 0 0;1 1 -1;1 -1 -1]
 y = [0 0 0;5 5 5;6 6 6]
 z = [0 0 0 ;1 1 -1;1 1 1]
 v = rand(3)
 fill(x,y,z)
 contour(x)
 contour(x,y,z)
```

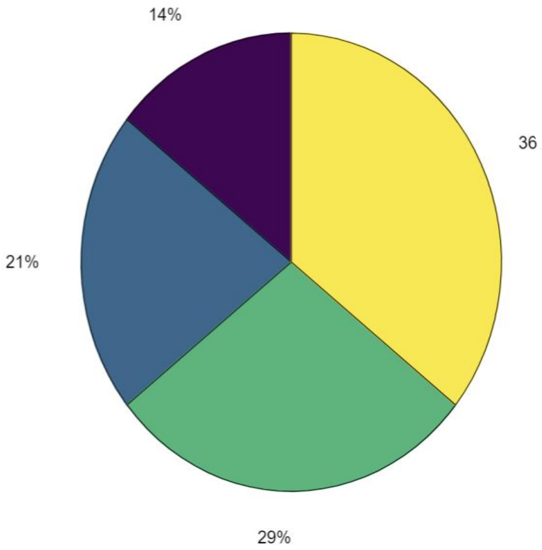
OBSERVATIONS:

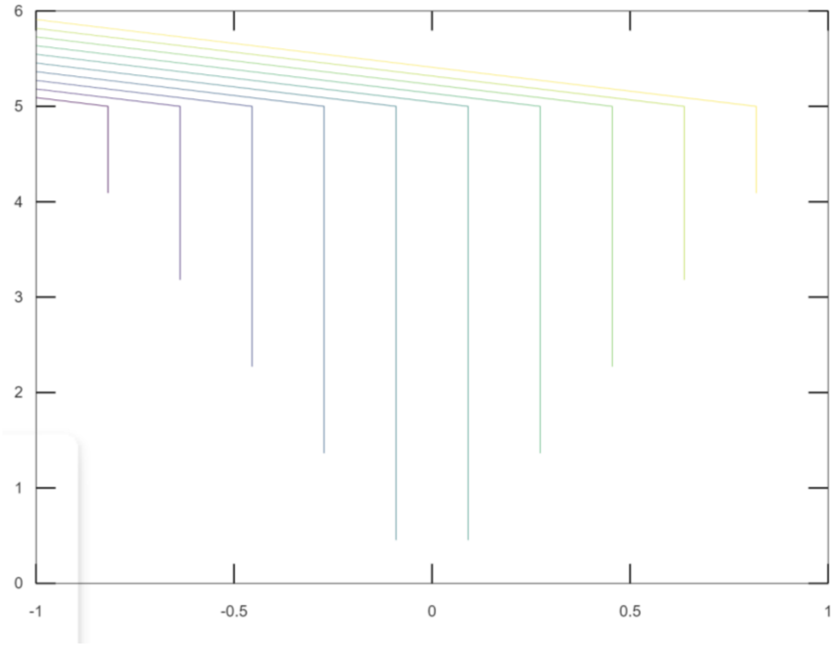


```
x =

   0   0   0
   1   1  -1
   1  -1  -1

y =

   0   0   0
   5   5   5
   6   6   6

z =

   0   0   0
   1   1  -1
   1   1   1

v =

   0.208459   0.572738   0.150935
   0.955272   0.857705   0.600597
   0.583652   0.968556   0.070720
```

RESULT:







DSCUSSION:

Basic 2D & 3D plots have been plotted , bar and pie charts have also been plotted.

Criteria	Total Marks	Marks Obtained	Comments
Concept (a)	2		
Implementation (b)	2		
Performance (c)	2		
Total	6		