Lab File

Basic Simulation Lab

(ES204)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Submitted to:

Dr Shalini Shah

Ast. Professor

CSE Department, ASET

Submitted by:

Shaina Mehta

A2305219268

B.tech. C.S.E.

3CSE-4Y

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

AMITY UNIVERSITY UTTAR PRADESH

NOIDA -201301

Exp No	Assignment Category	Code	Name of Experiment	Date of Allotment	Date of Evaluation	Max Marks	Marks Obtained	Faculty Sign
1	Mandatory Experiment		Creating a One – Dimensional Array (Row/Column), Creating a Two -Dimensional Array (Matrix of Given Size) and: (A) Performing Arithmetic Operations - Addition, Subtraction, Multiplication and Exponentiation. (B) Performing Matrix operations – Inverse, Transpose, Rank.	17-07-2020	14-08-2020			
2			Performing Matrix Manipulation – Concatenating, Indexing, Sorting, Shifting, Reshaping, Resizing and Flipping about Vertical Axis/ Horizontal Axis; Creating Arrays X and Y of given size (1XN) and performing: (A) Relational Operation ==, <=, >=, ~=. (B) Logical Operations ~, &, |, XOR.	24-08-2020	14-08-2020			
3			Generating a set of commands on a given Vector. Also generating a Random Sequence using rand()/randn()/randi()/randperm() and plotting them. (A) Adding up the values of the elements (check with Sum) (B) Computing the running sequence of them (check with Sum), where running Sum for elements j = the sum of the elements from 1 to j, inclusive.	3-07-2020	14-08-2020			
4			Evaluating a given expression and rounding it to the nearest integer value using round, floor, ceil and fix function. 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.	4-08-2020	14-08-2020			
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 suitable curves)on Rectangular Plot.	14-08-2020	11-09-2020			
6			Generating a sinusoidal signal of a given frequency (say 100Hz) and plotting with	21-08-2020	11-09-2020			

			graphical enhancements- Titling, Labelling, adding text, adding legends, adding new plots to existing plot, Plotting as multiple and subplot.					
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 =(0.45T+900) for T>100 Exercise: Testing the scripts written using (A) T=5 and h=-5 (B) T=110 and h=949.5	11-09-2020	26-09-2020			
9			Generating a square wave from sum of sine waves of certain amplitude and frequencies. The Gibbs phenomenon involves both the fact that Fourier sums overshoot at a jump discontinuity, and that this overshoot does not die out as the frequency increases $\sin(x) + \frac{1}{3}\sin(3x) + \frac{1}{5}\sin(5x) \ldots$	18-09-2020	26-09-2020			
9			Solving First Order Ordinary Differential Equation using Built-in Functions.	26-09-2020	23-10-2020			
10			Basic 2D and 3D plots: parametric space curve, polygons with vertices, 3D contour lines, pie and bar charts.	9-10-2020	23-102020			
	Viva	Viva						

Experiment 1

Aim: Creating a One – Dimensional Array (Row/Column), Creating a Two -Dimensional Array (Matrix of Given Size) and:

(A) Performing Arithmetic Operations -Addition, Subtraction, Multiplication and Exponentiation.
(B) Performing Matrix operations – Inverse, Transpose, Rank.

Tool Used: MATLAB 7.0 / Octave Online.

Theory: MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. It is a high-performance language for technical computing. It integrates, computation, visualisation, and programming in an easy to use environment where problems and solutions are expressed in a familiar mathematical notation. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages. It is used in Math and computation, Algorithm development, Data acquisition, Modelling, simulation and prototyping, Data analytics and, exploration, and visulisation, Scientific and engineering graphics, Application development includes 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.

Advantages of MATLAB:

MATLAB allows you to:

- Implement and test your algorithms easily.
- Develop the computational codes easily.
- Debug easily.
- Use a large database of built in algorithms.
- Process still images and create simulation videos easily.
- Symbolic computation can be easily done.
- Call external libraries.
- Perform extensive data analysis and visualization.
- Develop application with graphics user interface.

Disadvantages of MATLAB:

MATLAB has following disadvantages:

- Cross-compiling of MATLAB code to other language is difficult and requires a deep MATLAB knowledge to deal with errors produced.
- MATLAB is used mainly for scientific research and not suitable for development activities which are user specific.
- MATLAB is an interpreted language thus it can be slow.

- Poor programming practices can contribute to making MATLAB unacceptably slow.
- MATLAB is more expensive. The license is very costly, and users need to buy each module and need to pay for the same.
- MATLAB is not known to create application deployment for installation like task done by others which include the setting of files another executable which copies during installation.

Basic Theory of Experiment:

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

Consi5der two matrixes A and B is an m x n matrix and b is a n x p matrix, they could be multiplied together to produce an m x n matrix C. Matrix multiplication is a possibly only if the number of columns n in A is equal to the number of rows and columns in the second matrix. Each element in the $(i,j)^{th}$ position , in the resulting matrix C, is the summation of the products of elements in the i^{th} row of the first matrix with the corresponding element in the j^{th} column of the second matrix. In MATLAB, the 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 matrix is given by inv(A).

Transpose operations switches the rows and columns in a matrix. It is represented by a singular quotes (').

Rank function provides an estimate of the number of linearly independent rows and columns k=rank(A,tol) returns the number of the singular values of A that are larger than tol.

MATLAB Program and Output:

(A) % Algebric operatons in a matrix%

%Addition and Subtraction%

Input>A=[1 2 3 4 5]

display(A);

B=[1 2 4 7 7]

display(B);

C=A+B;

display(C);

D=A-B;

display(D);

Output>

A =

1 2 3 4 5

B =

1 2 4 7 7

C =

2 4 7 11 12

D =

0 0 -1 -3 -2

%Multiplication of two matrices%

Input>A=[1 2 3 4 5];

B=[1 2 3 7 6];

display(A);

display(B);

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

display(D);

E=[1 2 3;2 3 4;3 4 5];

display(E);

F=D*E;

display(F);

G=[1 2 3 4 5];

display(G);

H=2*G;

display(H);

Output>

A =

1 2 3 4 5

B =

1 2 3 7 6

D =

1 2 3

4 5 6

7 8 9

E =

1 2 3

2 3 4

3 4 5

F =

14 20 26

32 47 62

50 74 98

G =

1 2 3 4 5

H =

2 4 6 8 10

(B) %Matrix Operations%

Input>A=[1 2 3;45 6 7;7 89 3];

display(A);

B=inv(A);

display(B);

C=rank(A);

display (C);

D=A';

display(D);

Output>

A =

1 2 3

45 6 7

7 89 3

B =

-5.4446e-02 -2.3488e-02 -3.5997e-04

-7.7394e-03 -1.6199e-03 -1.1519e-02

-3.5664e-01   -6.7495e-03   -7.5594e-03

C = 3

D =

1   45 7

2   3   89

3   7   3

Result: The matrix operation has been performed successfully.

Experiment 2

Aim: Performing Matrix Manipulation – Concatenating, Indexing, Sorting, Shifting, Reshaping, Resizing and Flipping about Vertical Axis/ Horizontal Axis; Creating Arrays X and Y of given size (1XN) and performing:

(A) Relational Operation ==, <=, >=, ~=.
(B) Logical Operations ~, &, |, XOR.

Tools Used: MATLAB 7.0/Octave Online.

Theory: An operator is a symbol that tells the compiler to perform certain mathematical and logical manipulations. MATLAB is designed to operate primarily o n whole matrices and arrays. MATLAB allows four types of elementary operations:

- Arithmetic Operations
- Logical Operations
- Relational Operations
- Bitwise Operations
- Set Operations

Arithmetic Operations: MATLAB allows two different types of arithmetic operations that are:

- Matrix Arithmetic Operations
- Array Arithmetic Operations

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

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 the both cases. The following table gives the brief description of the operators:

Operator	Description
+	Addition and unary plus. A+B adds 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 B from A. A and B must have the same size, as one is scalar. A scalar can be subtracted from the matrix of any size.
*	Matrix multiplication. C=A*B is the linear algebraic product of the matrices A and B. For a non-scalar matrix, the no of columns A must be equal to the no of rows of B. A scalar can multiply the matrix of any size.
.*	Array multiplication. A.*B is the element by the element of the product of arrays A and B. A must have the same size as one is scalar.
/	Backslash or matrix left division. If A and B is the square matrix then A and B as the inv(A)*B , except it is computed in a different way. If A is a n by n matrix and Bis a column vector with n components or the matrix with such columns, then X=A/B is the solution to the equation AX=B. A warning message will display if A is badly scalar or nearly scalar.

./	Array left division. A./B is the matrix with the elements B(i,j) and A(I,j). A and B must have the same size, unless one of them is scalar.
^	Matrix power. X^p is X is the power of p, if p is the scalar. If p is an integer then power computed by repeating squaring. If the integer is negative then, X is inverted first. For other values of p, the calculation involves eigen vectors and eigen values such that if [V,D]=eig(X), then X^p=V*D.^p/V.

Relational Operators: It can also work for both scalar and non scalar data. Relational operators perform element by element comparisons on the two arrays and return the logical array of the same size where the element is to be logical (1) true where it is true and else logical (0) false where the element is false. The following element shows the relational operator available in the MATLAB:

Operator	Description
==	Equal To
~=	Not Equal To
>=	Greater Than Equal To
<=	Less Than Equal To
>	Greater Than
<	Less Than

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

- Element-wise: These operators operate on corresponding elements of logical arrays. In other words we can say, these operators operate element-by-element on logical arrays. The symbols &, |, and ~ are the logical array operators AND, OR, and NOT.
- Short-circuit: these operators operate on scalar, logical expressions. Short-circuit logical operators allow short-circuiting on scalar logical expressions and operations. The symbols && and || are the logical short-circuit operators AND and OR.

Bitwise Operator: Bitwise operator works on bits and performs bit-by-bit operation. MATLAB provides various functions for bit-wise operations like 'bitwise and', 'bitwise or' and 'bitwise not' operations, shift operation, etc.

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

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

Concatenating Matrices: In MATLAB one can concatenate two matrices to create a larger matrix. The pair of square brackets '[ ]' is the concatenation operator. MATLAB allows two types of concatenations −

- Horizontal concatenation
- Vertical concatenation

When you concatenate two matrices by separating those using commas, they are just appended horizontally. It is called horizontal concatenation. Alternatively, if you concatenate two matrices by separating those using semicolons, they are appended vertically. It is called vertical concatenation.

MATLAB Programs and Outputs:

(A) %Concatenation of Two Arrays%

Input> A=[6 4 3 ;6 2 3;9 2 1];

display(A);

B=ones(3)*8;

display(B);

C=[A;B];

display(C);

Output>

A =

6 4 3

6 2 3

9 2 1

B =

8 8 8

8 8 8

8 8 8

C =

6 4 3

6 2 3

9 2 1

8 8 8

8 8 8

8 8 8

(B) %To print the magic matrix, random matrix, and pascal of the matrix%

```
octave:2> A=magic(4);
octave:3> display(A);
A =
  16   2   3  13
   5  11  10   8
   9   7   6  12
   4  14  15   1
octave:4> B=pascal(5);
octave:5> display(5);
5
octave:6> display(B);
B =
   1   1   1   1   1
   1   2   3   4   5
   1   3   6  10  15
   1   4  10  20  35
   1   5  15  35  70
octave:7> C=rand(6);
octave:8> display(C);
C =
  0.610641  0.592547  0.024331  0.301627  0.462425  0.299010
  0.774514  0.750017  0.359707  0.906125  0.597837  0.262426
  0.206184  0.583777  0.197114  0.111088  0.056178  0.837177
  0.307490  0.435049  0.451051  0.811341  0.992942  0.645485
  0.926348  0.499336  0.566914  0.607301  0.876838  0.755639
  0.982158  0.041164  0.524166  0.453790  0.849815  0.308016
```

(C) %To concatenate two strings using MATLAB%

Input> s1="engfinering";

```
s2="science";
s3=strcat(s1,s2);
display(s3);
s4="courses for";
 s5=" engineering";
s6=strcat(s4,s5);
display(s6);
```

Output>

s3 = engfineringscience

s6 = courses for engineering

(D)%To sort the rows and columns of a matrix%

Input>

```
A=[2 3 1; 4 5 2 ;1 1 2];
display(A);
C=sort(A);
display(C);
D=sort(A,'descend');
display(D);
E=sort(A,'ascend');
display(E);
F=sortrows(2);
display(F);
F=sortrows(A,2);
display(F);
```

Output>

A =

2 3 1

4 5 2

1 1 2

C =

1 1 1

2 3 2

4 5 2

D =

4 5 2

2 3 2

1 1 1

E =

1 1 1

2 3 2

4 5 2

F = 2

F =

1 1 2

2 3 1

4 5 2

(E) %To rotate, resize and reshape the matrix%

Input>A=[2 3 4 5;1 1 2 4;6 7 8 9;5 6 7 8;9 7 4 0];

display(A);

B=rot90(A);

display(B);

C=reshape(A,5,4);

display(C);

D=reshape(A,4,5);

display(D);

Output>

A =

2 3 4 5

1 1 2 4

6 7 8 9

5 6 7 8

9 7 4 0

B =

5 4 9 8 0

4 2 8 7 4

3 1 7 6 7

2 1 6 5 9

C =

2 3 4 5

1 1 2 4

6 7 8 9

5 6 7 8

9 7 4 0

D =

2 9 6 8 4

1 3 7 7 9

6 1 4 4 8

5 7 2 5 0

(F) %To flip the rows and columns of a matrix%

Input>A=[2 3 4;8 7 6;4 5 6;4 5 3];
display(A);
B=fliplr(A);
display(A);
display(B);

Output>
A =
2 3 4
8 7 6
4 5 6
4 5 3
A =
2 3 4
8 7 6
4 5 6
4 5 3

B =
4 3 2

```
  6   7   8
  6   5   4
  3   5   4
```

(G)%To perform logical and relational operations in a matrix%

```
Input>X=[1 2 3 4];
Y=[3 4 56];
display(X);
display(Y);
Y=[1 4 5 6];
display(Y);
display(X>Y);
display(X<Y);
Y=[2 3 7 5];
display(Y);
display(X>Y);
display(X<Y);
display(X==Y);
display(X~=Y);
X2=and(X,Y);
display(X2);
X3=or(X,Y);
display(X3);
X4=xor(X,Y);
display(X4);
```

Output>

```
X =
  1   2   3   4
Y =
   3    4   56
Y =
  1   4   5   6
```

0 0 0 0

0 1 1 1

Y =

2 3 7 5

0 0 0 0

1 1 1 1

0 0 0 0

1 1 1 1

X2 =

1 1 1 1

X3 =

1 1 1 1

X4 =

0 0 0 0

Result: The matrix and array operations have been performed successfully.

Experiment 3

Aim: Generating a set of commands on a given Vector. Also generating a Random Sequence using rand()/randn()/randi()/randperm() and plotting them.

(A) Adding up the values of the elements (check with Sum)
(B) Computing the running sequence of them (check with Sum), where running Sum for elements j = the sum of the elements from 1 to j, inclusive.

Tools Used: MATLAB7.0/Octave Online.

Theory: When you create random numbers using software, the results are not random in a strict, mathematical sense. However, software applications, such as MATLAB®, algorithms that make your results appear to be random and independent. The results also pass various statistical tests of randomness and independence. This apparently random and independent numbers are often described as pseudorandom and pseudo independent. You can use these as if they are truly random and independent. One benefit of using pseudorandom, pseudoindependent numbers is that you can repeat you can repeat a random number calculation at any time. This can be useful in testing or diagnostic situations. All the random number functions, rand, randi, randn and randperm draw values from a shared random number generator. Every time you start MATLAB generator resets itself to the same state. Therefore, a command such as rand(2,2) returns the same result anytime you execute it immediately following startup. Also, any script or function that calls a random number functions return the same result whenever you restart. There are four fundamental random number functions: rand, randi, randn and randperm.

Rand(): The rand function returns real numbers between 0 and 1 that are drawn from a uniform distribution. For example:

r1= rand(1000,1);

r1 is a 1000 – by – 1 column vector containing real floating – point numbers drawn from a uniform distribution. All the values in r1 are in the open interval, (0,1). A histogram of these values is roughly flat, which indicated a fairly uniform sampling of numbers.

Randi():

r2= randi(10,1000,1);

r2 is a 1000 – by – 1 column vector containing integer values drawn from a discrete uniform distribution whose range is 1,2…,10. A histogram is roughly flat, which indicates a fairly uniform sampling of integer between 1 and 10.

Randn(): The randn function returns arrays of real floating – point numbers that are drawn from a standard normal distribution. For example:

r3=randn(1000,1);

r3 is a 1000 – by – 1 column vector containing numbers drawn from a standard normal distribution. A histogram of r3 looks a roughly normal distribution whose mean is 0 and standard deviation is 1.

Randperm(): The randperm function is used to create arrays of random integer values that have no repeated values. For example:

r4= randperm(15,5);

r4 is a 1 - by – 5 array containing randomly selected integer values on the closed interval, [1,15]. Unlike randi, which can return an array containing repeated values. The array returned by randperm has no repeated values. Successive calls to any of these functions return different results. This behaviour is useful for creating several different arrays of random values.

Some More Functions:

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

- A=sum(X);

  It is used to find the sum of all the elements of the matrix X.

- B=sum(X(2:4));

  It is used to find the sum of elements from the 2^{nd} element to the 4^{th} element of the matrix X.

- C=cumsum(Y,1);

  It is used to find the cumulative sum of the columns of the matrix Y.

- D=cumsum(Y,2);

  It is used to find the cumulative sum of the rows of the matrix Y.

- E=sum(Y,1);

  It is used to find the sum of the columns of the matrix Y.

- F=sum(Y,2);

  It is used to find the sum of the rows of the matrix Y.

MATLAB Programs and Output:

(A) %To find out the random matrix and plot its graph%

Input>A=rand(5,5);

display(A);

plot(A);

title('Random function 1');

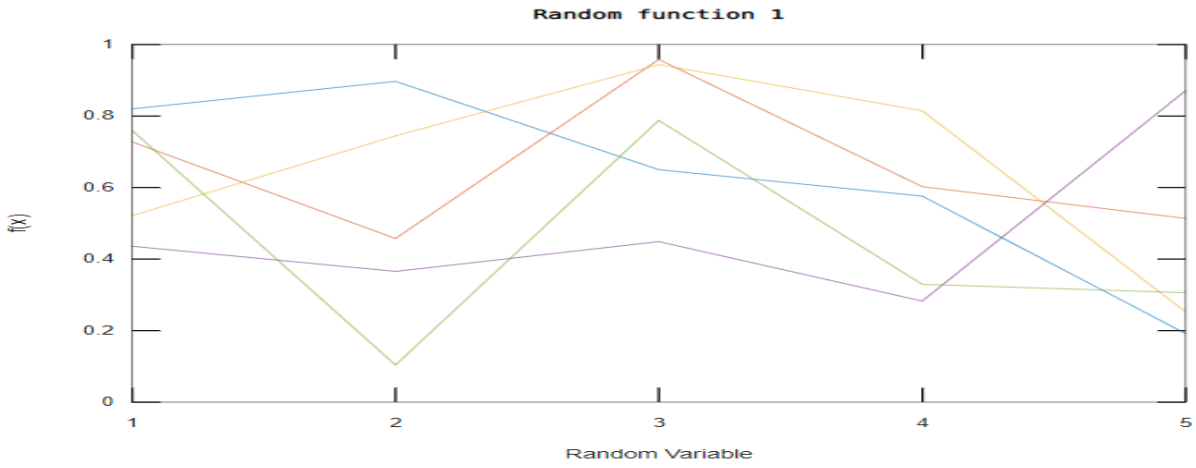
xlabel('Random Variable');

ylabel('f(x)');

```
B=randi(34,6,4);
display(B);
plot(B);
title('Random function 2');
xlabel('Random Variable');
ylabel('f(x)');
C=randn(4,7);
display(C);
plot(C);
title('Random Function 3');
xlabel('Random Variable');
ylabel('f(x)');
D=randperm(34,7);
display(D);
plot(D);
title('Random Function 4');
xlabel('Random Variables');
ylabel('f(x)');
```

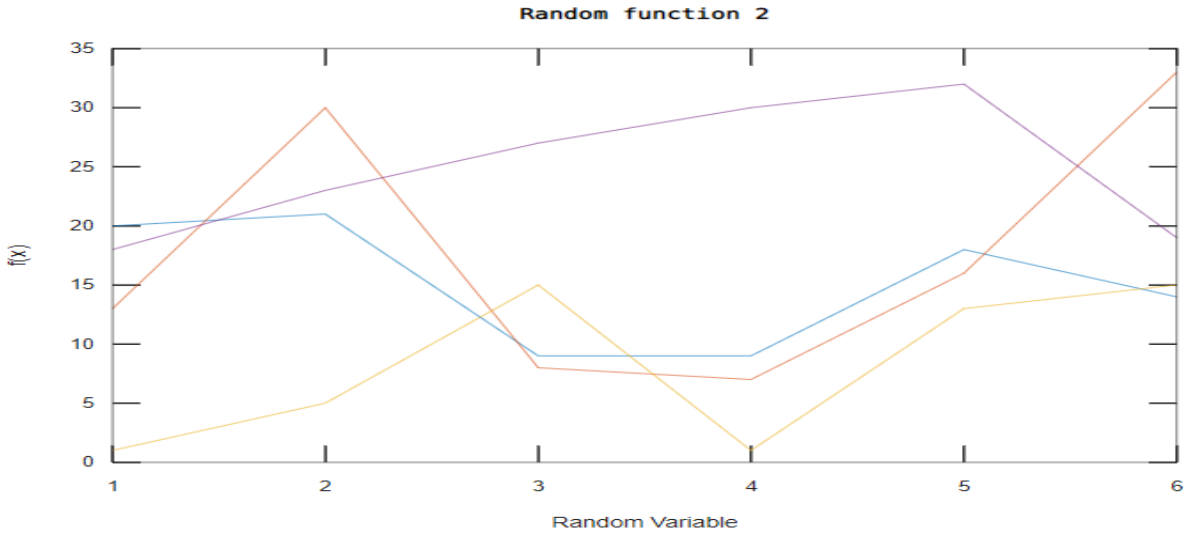
Output>

```
A =
   0.8194   0.7275   0.5210   0.4357   0.7597
   0.8966   0.4573   0.7442   0.3653   0.1038
   0.6499   0.9579   0.9435   0.4485   0.7875
   0.5758   0.6023   0.8143   0.2825   0.3291
   0.1918   0.5136   0.2524   0.8707   0.3059
```



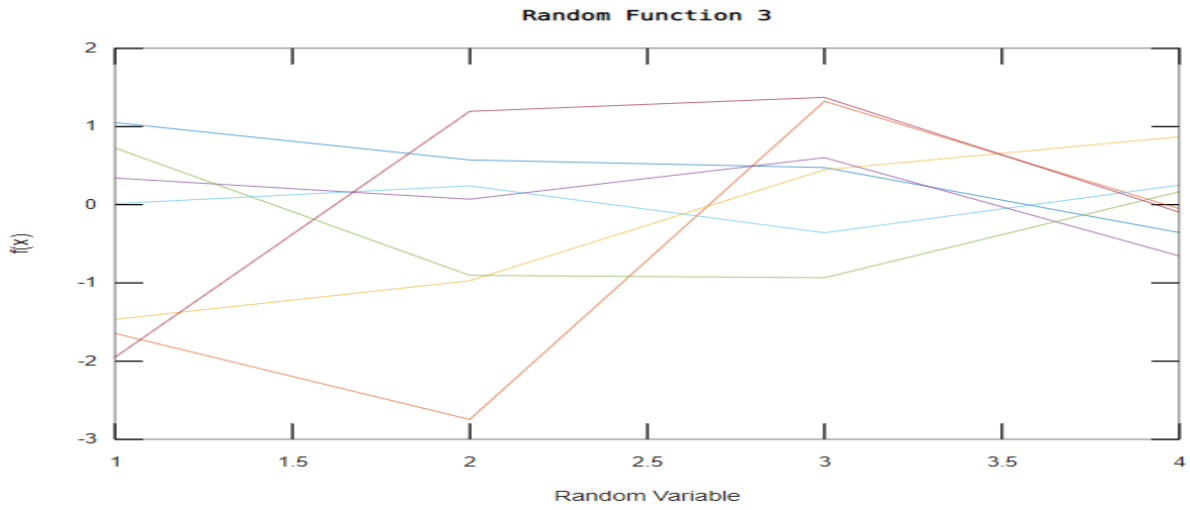
B =

20	13	1	18
21	30	5	23
9	8	15	27
9	7	1	30
18	16	13	32
14	33	15	19



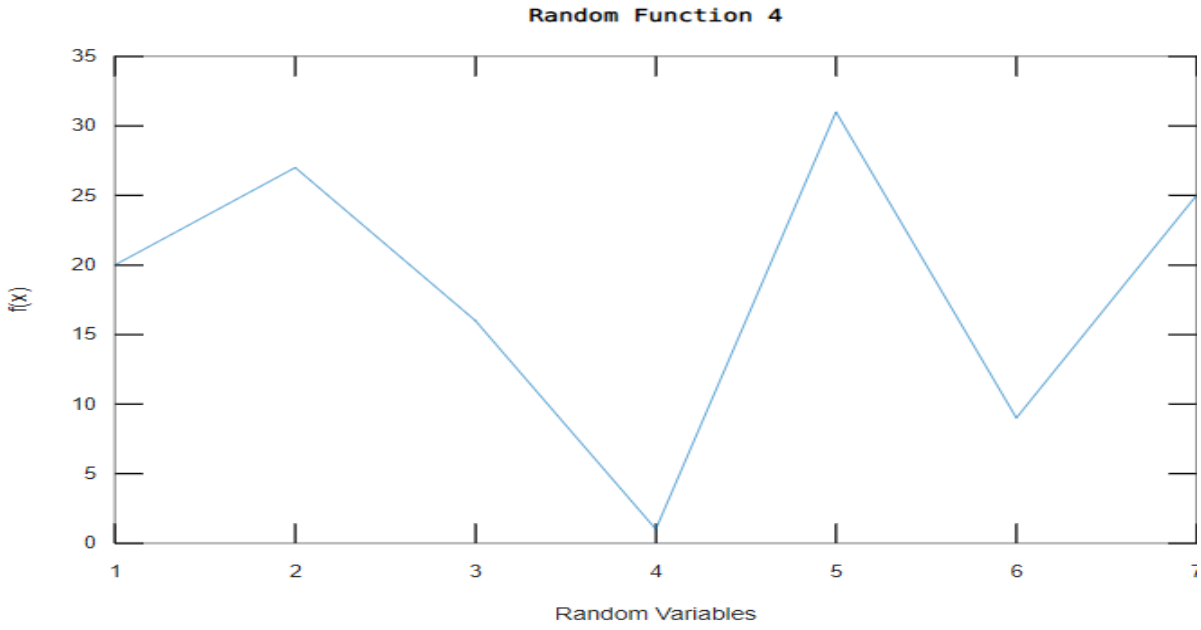
C =

1.050079	-1.645146	-1.467225	0.340486	0.724462	0.014039	-1.951149
0.571948	-2.746159	-0.973175	0.070260	-0.903433	0.240191	1.192927
0.474559	1.322453	0.446385	0.601509	-0.933585	-0.358949	1.372020
-0.356053	-0.050788	0.869064	-0.654631	0.165355	0.250016	-0.095952



D =

20 27 16 1 31 9 25



(B) % To find out the sum of elements of the matrix, its rows and columns and also find out the cumulative sum of the rows and columns of the matrix%

Input>X=[1 2 3 3 2 12 3];

display(X);

Y=sum(X);

display(Y);

Z=sum(X(1:5));

display(Z);

Z = 11

```
W=sum(X(3:5));
display(Z);
display(W);
A=magic(3);
display(A);
B=cumsum(A,1);
display(B);
C=cumsum(A,2);
display(C);
D=sum(A,1);
display(D);
E=sum(A,2);
display(E);
```

Output>

```
X=
    1    2  3  3  2  12  3
Y = 26
Z = 11
W = 8
A =
   8   1   6
   3   5   7
   4   9   2
B =
    8    1    6
   11    6   13
   15   15   15
C =
    8    9   15
    3    8   15
```

```
   4   13   15
```

D =

```
   15   15   15
```

E =

```
   15
   15
   15
```

Result: The matrix and vector operations has been performed successively.

Experiment 4

Aim: Evaluating a given expression and rounding it to the nearest integer value using round, floor, ceil and fix function. Also generating and plots of:

(C) Trigonometric functions - sin(t), cos(t), tan(t), sec(t), cosec(t) and cot(t) for a given duration t.
(D) Logarithmic and other functions – log(A), log10(A), square root of A, real nth root of A.

Tools Used: MATLAB 7.0.

Theory:

Ceil():

Y=ceil(A);

- Rounds the elements of a to the newest integers greater than or equal to A.
- For complex A, the imaginary and real parts are rounded independently.

Round():

Y=round(X);

- Rounds the elements of X of the nearest integers.
- For complex X, the imaginary and real parts are rounded independently.

Floor():

Y=floor(A);

- Rounds the elements of A to the nearest integers less than or equal to A.
- For complex A, the imaginary and real parts are rounded independently.

Fix():

Y=fix(A);

- Rounds the elements of A towards zero, resulting in an array of integers.
- For complex A, the imaginary and real parts are rounded independently.

Sin():

Y = sin(X)

- Returns the sine of the elements of X. The sin function operates element - wise on arrays. The function accepts both real and complex inputs.
- For real values of X, sin(X) returns real values in the interval [-1, 1].
- For complex values of X, sin(X) returns complex values.

Cos():

Y = cos(X)

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

Tan():

Y = tan(X);

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

Sec():

Y = sec(X);

- Returns the secant of the elements of X. The sec function operates element - wise on arrays. The function accepts both real and complex inputs.
- For real values of X, sec(X) returns real values in the interval [-∞, -1] and [1, ∞].
- For complex values of X, sec(X) returns complex values.

Csc():

Y = csc(X);

- Returns the cosecant of the elements of X. The csc function operates element - wise on arrays. The function accepts both real and complex inputs.
- For real values of X, csc(X) returns real values in the interval [-∞, -1] and [1, ∞].
- For complex values of X, csc(X) returns complex values.

Cot():

Y = cot(X);

- Returns the cotangent of elements of X. The cot function operates element - wise on arrays. The function accepts both real and complex inputs.
- For real values of X, cot(X) returns real values in the interval [-∞, ∞].
- For complex values of X, cot(X) returns complex values.

Log():

Y = log(X);

- Returns the natural logarithm ln(x) of each element in array X.
- The log function's domain includes negative and complex numbers, which can lead to unexpected results if used unintentionally. For negative and complex numbers z = u + i*w, the complex logarithm log(z) returns

log(abs(z)) + 1i*angle(z)

Log10():

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.

Log2(X):

Y = log2(X);

- Computes the base 2 logarithm of the elements of X such that $2^Y=X$.
- For Example:
  [F,E] = log2(X) returns arrays F and E such that $X=F\cdot 2^E$. The values in F are typically in the range 0.5 <= abs(F) < 1.

Exp(X):

Y = exp(X);

- Returns the exponential ex for each element in array X.
- For complex elements z = x + iy, it returns the complex exponential $e^z=e^x(cosy+isiny)$.

Sqrt(X):

Y = 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.
- The sqrt function's domain includes negative and complex numbers, which can lead to unexpected results if used unintentionally. For negative and complex numbers z = u + i*w, the complex square root sqrt(z) returns
  sqrt(r)*(cos(phi/2) + 1i*sin(phi/2))
  where r = abs(z) is the radius and phi = angle(z) is the phase angle on the closed interval -pi <= phi <= pi.

Nthroot():

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.

MATLAB Programs:

(A) % To round off the numbers to its nearest value of the matrix%

Input> A=[1.34,2,12,3.45,23.34,34.5+45.6i,23.56];

display(A);

B=ceil(A);

display(B);

C=round(A);

display(C);

D=fix(A);

display(D);

E=floor(A);

display(E);

Output>

A =

Columns 1 through 3:

1.3400 + 0i 2.0000 + 0i 12.0000 + 0i

Columns 4 through 6:

3.4500 + 0i 23.3400 + 0i 34.5000 + 45.6000i

Column 7:

23.5600 + 0i

B =

Columns 1 through 6:

2 + 0i 2 + 0i 12 + 0i 4 + 0i 24 + 0i 35 + 46i

Column 7:

24 + 0i

C =

Columns 1 through 6:

1 + 0i 2 + 0i 12 + 0i 3 + 0i 23 + 0i 35 + 46i

Column 7:

24 + 0i

D =

Columns 1 through 6:

1 + 0i 2 + 0i 12 + 0i 3 + 0i 23 + 0i 34 + 45i

Column 7:

23 + 0i

E =

Columns 1 through 6:

1 + 0i 2 + 0i 12 + 0i 3 + 0i 23 + 0i 34 + 45i

Column 7:

23 + 0i

(B) %To plot the graph of all the trigonometric functions%

%Sine Function%

Input>X=0:1:10*pi;

A=4;

Y=A*sin(X);

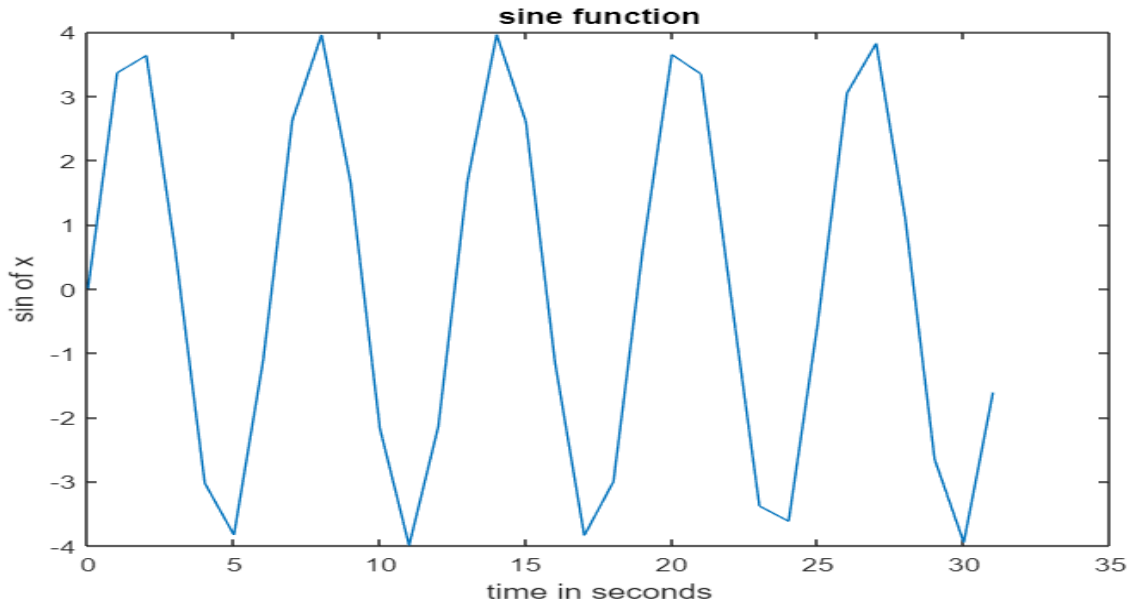
plot(X,Y);

title('sine function');

xlabel('time in seconds');

ylabel('sin of x');

Output>



%Cosine Function%

Input>X=0:1:10*pi;

A=4;

Y=A*cos(X);

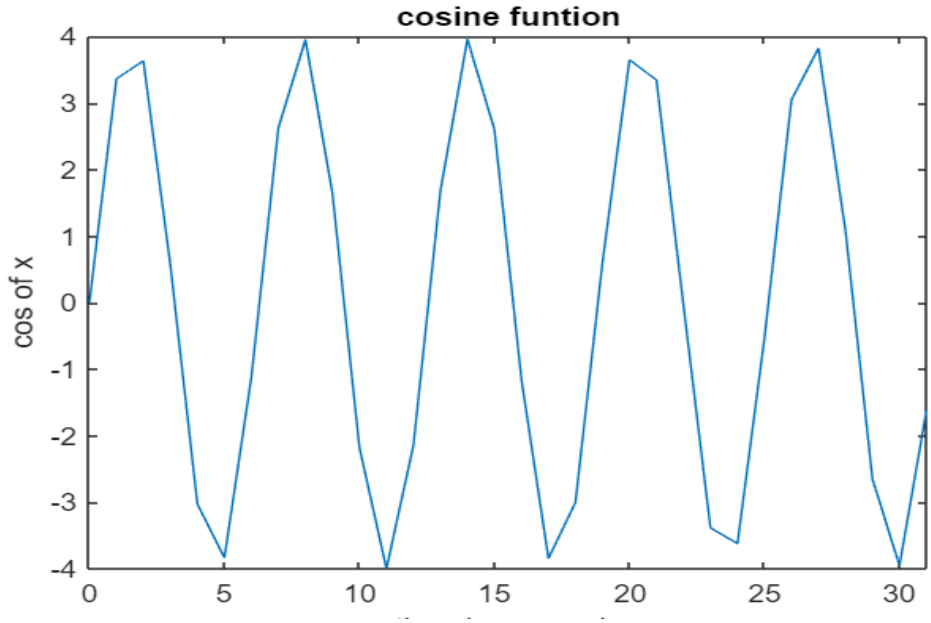
plot(X,Y);

title(‘cosine function’);

xlabel(‘time in seconds’);

ylabel(‘cos of x’);

Output>



%Tangent Function%

Input>X=0:1:10*pi;

A=4;

Y=A*tan(X);

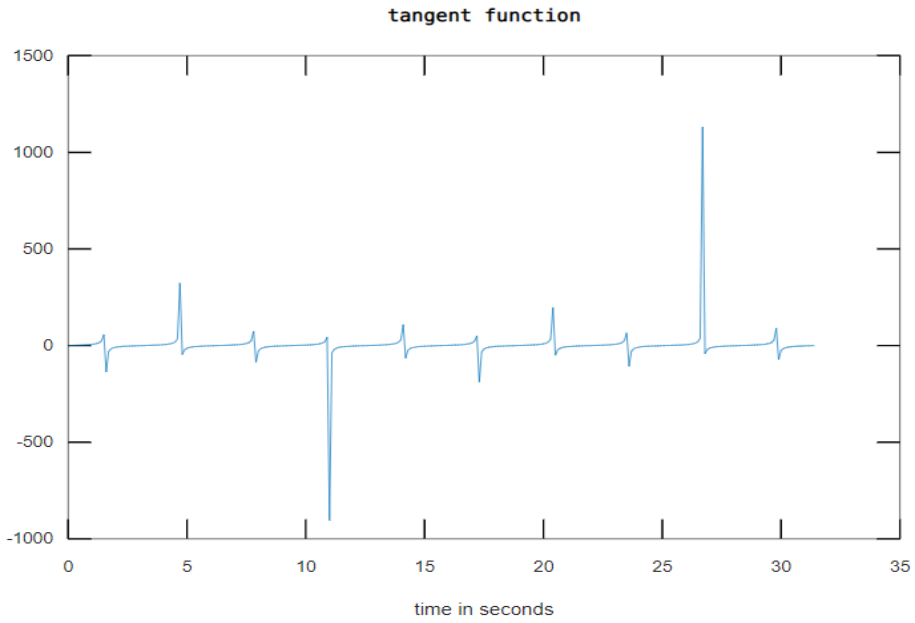
plot(X,Y);

title(‘tangent function’);

xlabel(‘time in seconds’);

ylabel(‘tan of x’);

Output>



%Secant Function%'

Input>X=0:0.01:10*pi;

A=4;

Y=A*sec(X);

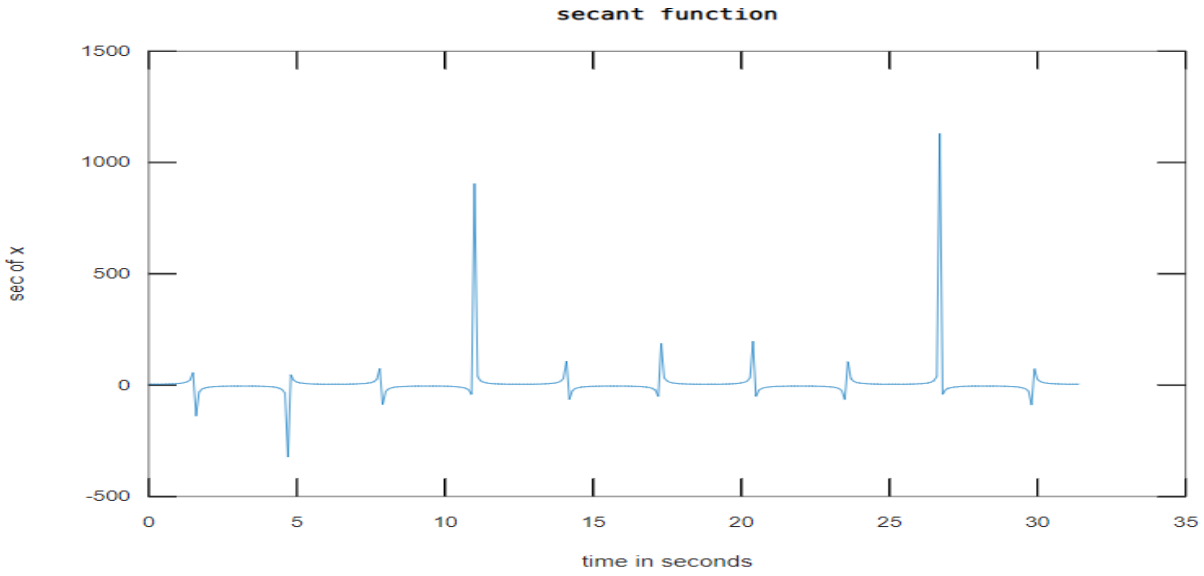
plot(X,Y);

title('secant function');

xlabel('time in seconds');

ylabel('sec of x');

Output>



%Cosecant Function%

Input>X=0:0.01:10*pi;

A=4;

Y=A*cosec(X);

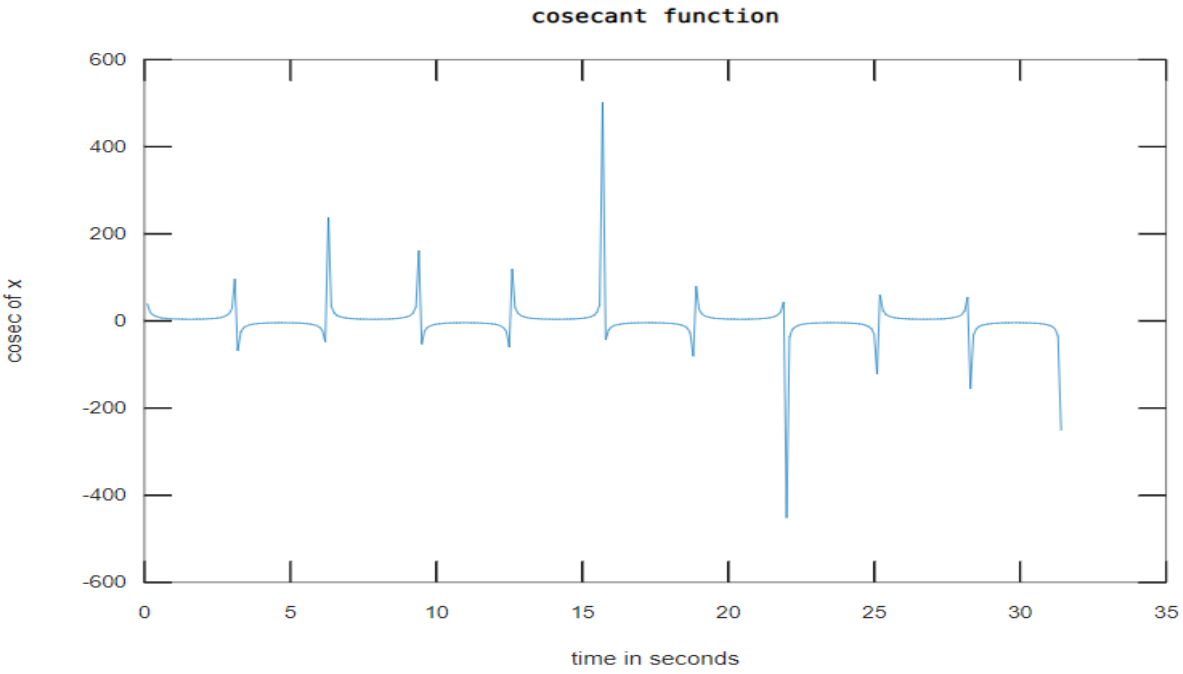
plot(X,Y);

title(‘cosecant function’);

xlabel(‘time in seconds’);

ylabel(‘cosecant of x’);

Output>



%Cotangent Function%

Input>X=0:0.01:10*pi;

A=4;

Y=A*cot(X);

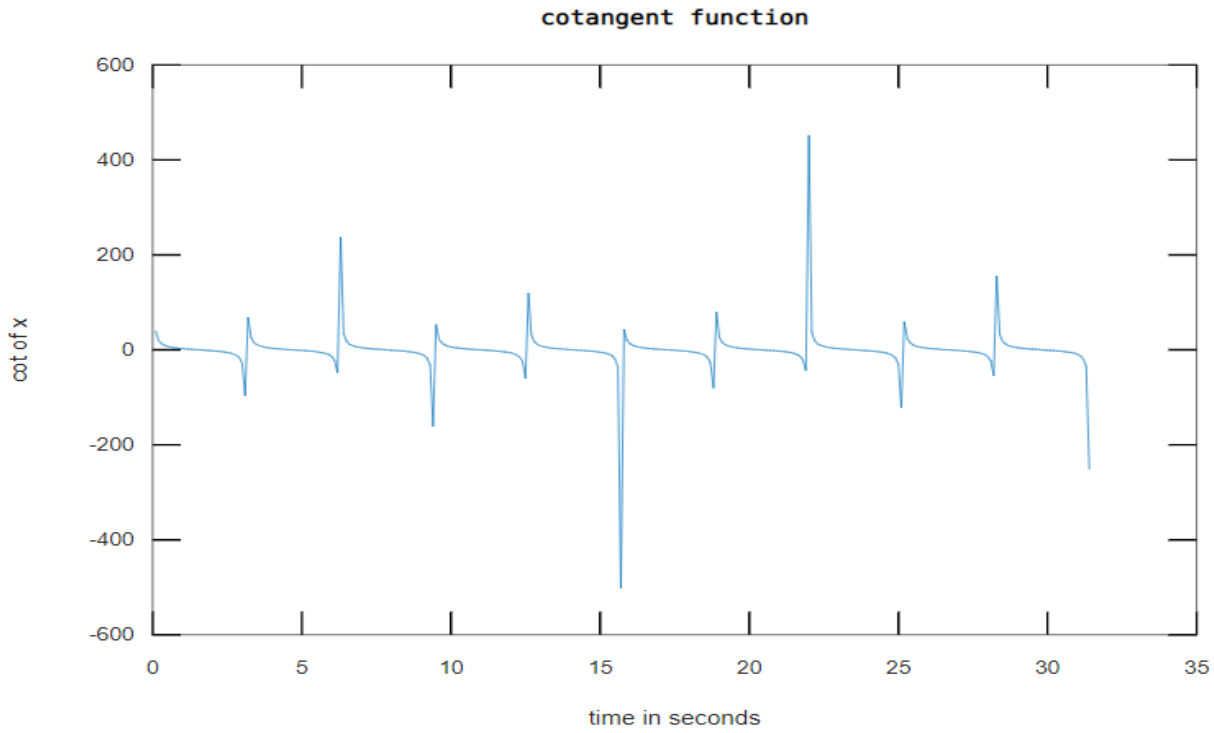
plot(X,Y);

title(‘cotangent function’);

xlabel(‘time in seconds’);

ylabel(‘cot of x’);

Output>



(C) %To plot the graph of logarithmic, exponential, square root and nth root of a number functions%

%Natural Log%

Input>X=0:1:100;

Y = log(X);

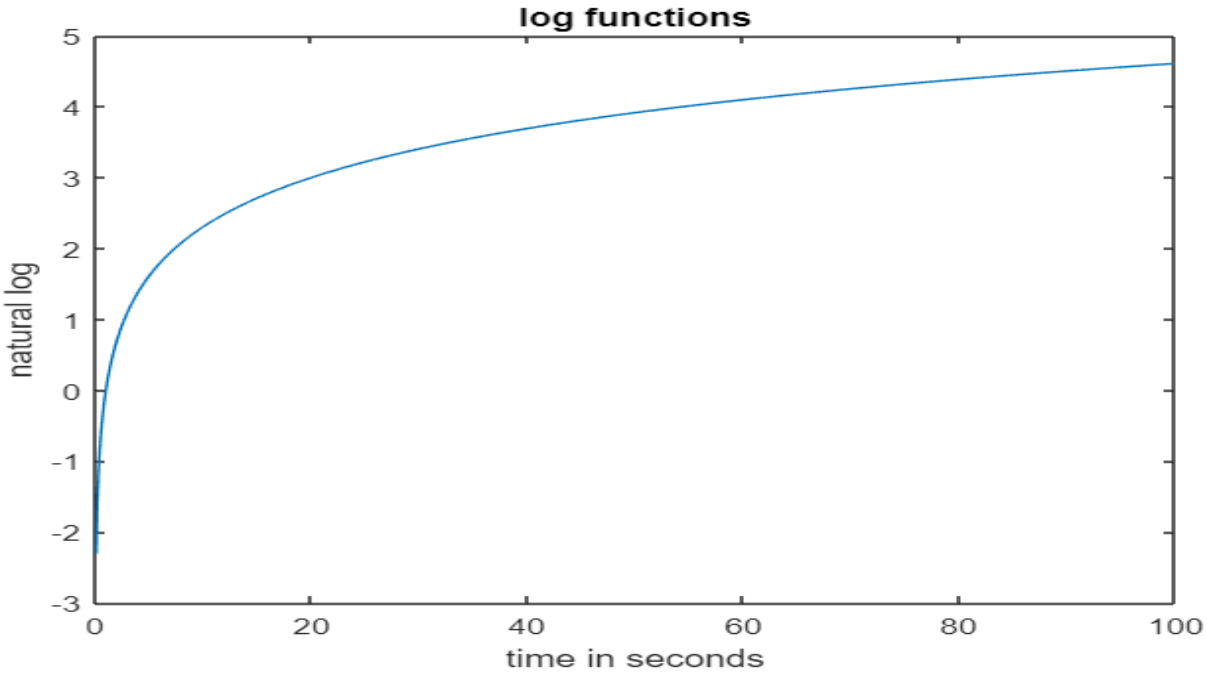
plot(X,Y);

title('log functions');

xlabel('time in seconds');

ylabel('natural log');

Output>



%Log Base 10%

Input>X=0:1:100;

Y = log10(X);

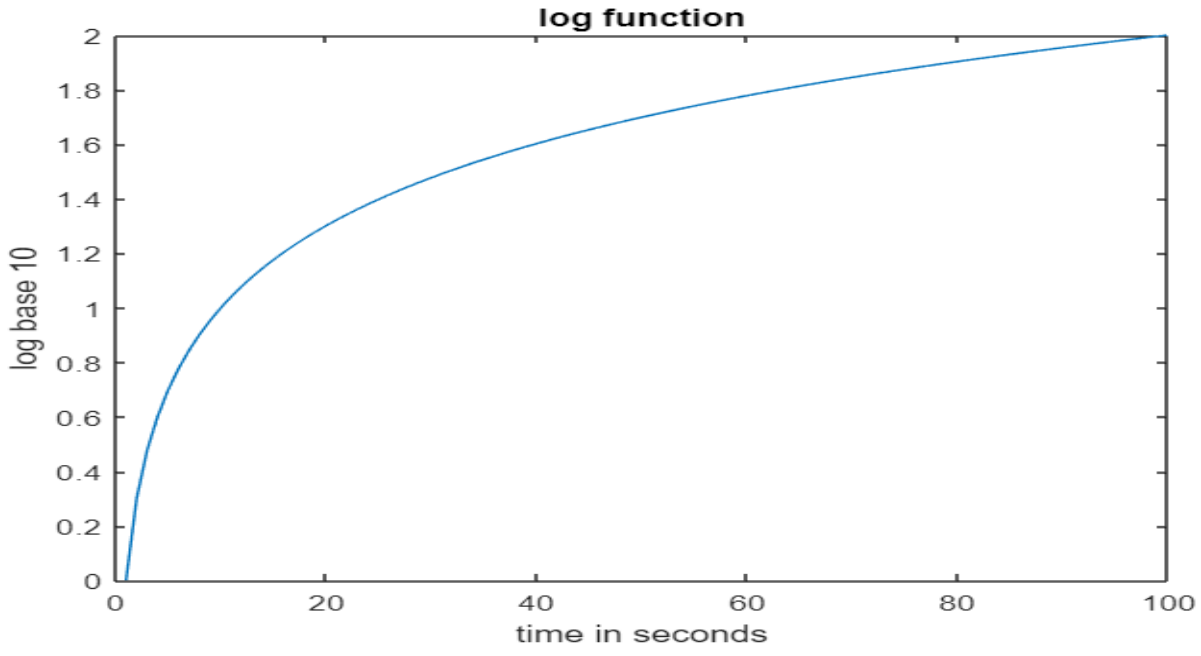
plot(X,Y);

title('log functions');

xlabel('time in seconds');

ylabel('log base 10');

Output>



%Log Base 2%

Input>X=0:1:100;

Y = log(X);

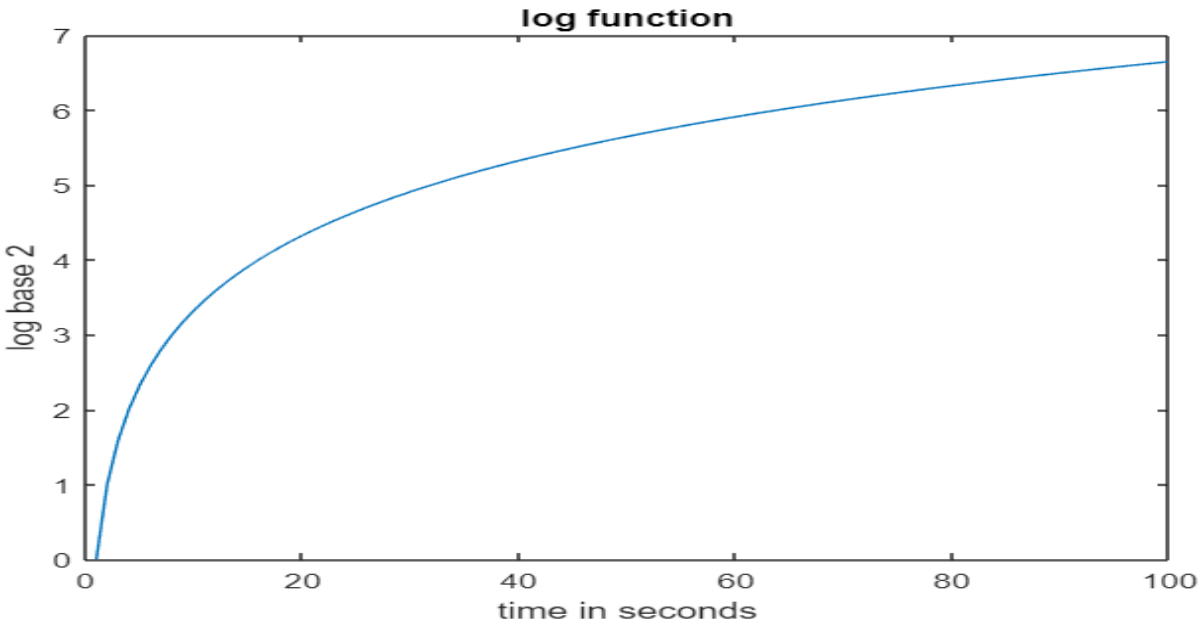
plot(X,Y);

title('log functions');

xlabel('time in seconds');

ylabel('log base 2');

Output>



%Exponential of X%

Input>X=0:1:100;

Y = exp(X);

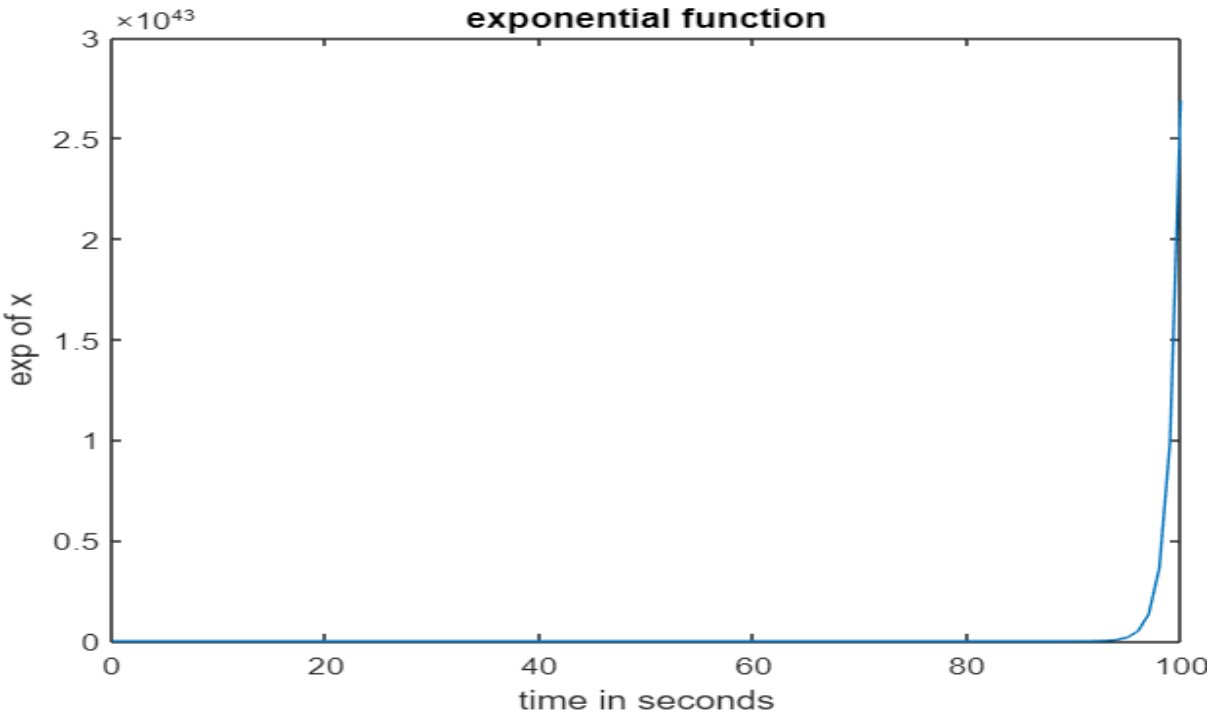
plot(X,Y);

title('exponential functions');

xlabel('time in seconds');

ylabel('exponential of X');

Output>



%Exponential of -X%

Input>X=0:1:100;

Y = exp(-X);

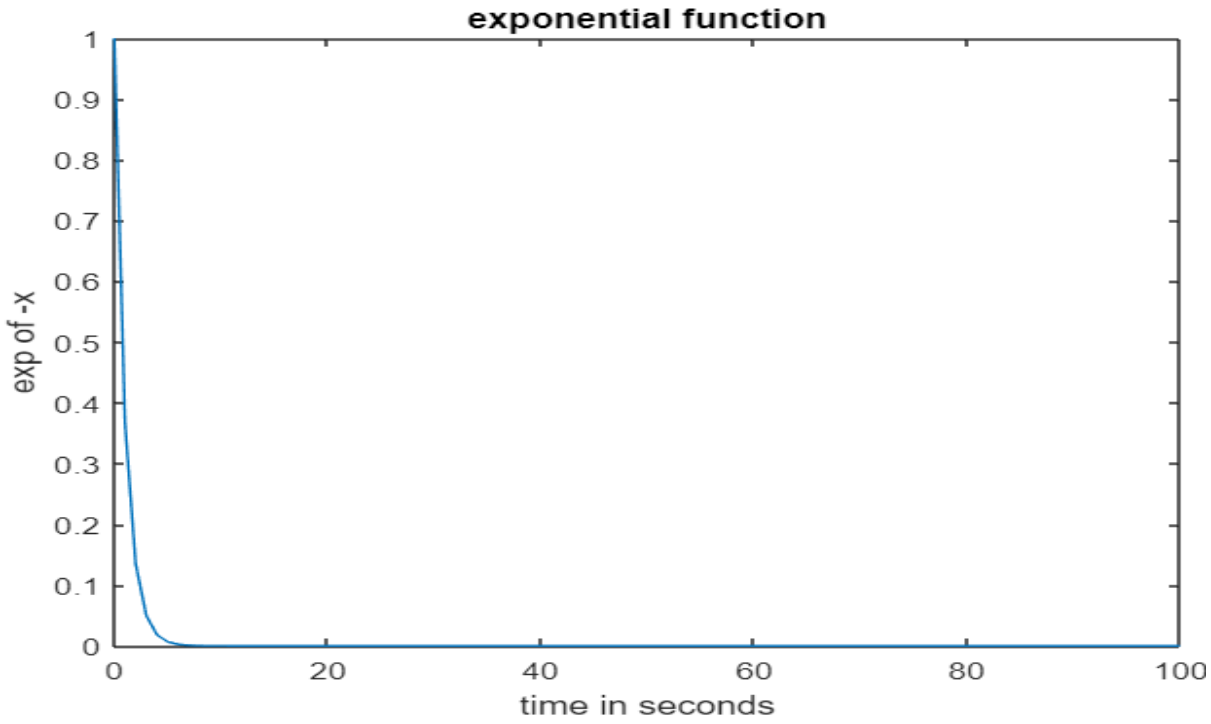
plot(X,Y);

title('exponential functions');

xlabel('time in seconds');

ylabel('exponential of -X');

Output>



%Square Root Function%

Input>X=0:1:100;

Y = sqrt(X);

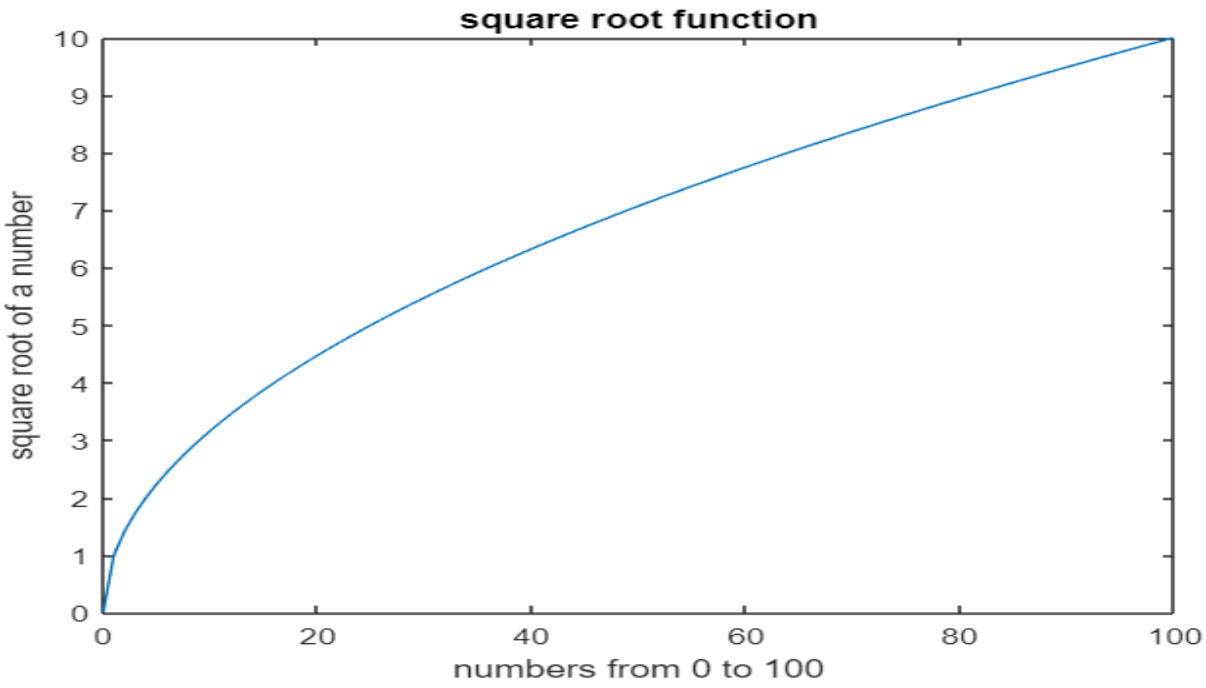
plot(X,Y);

title('square root function');

xlabel('numbers from 0 to 100');

ylabel('square root of a number');

Output>



%Nth Root Function%

Input>X=0:1:100;

Y = nthroot(X,4);

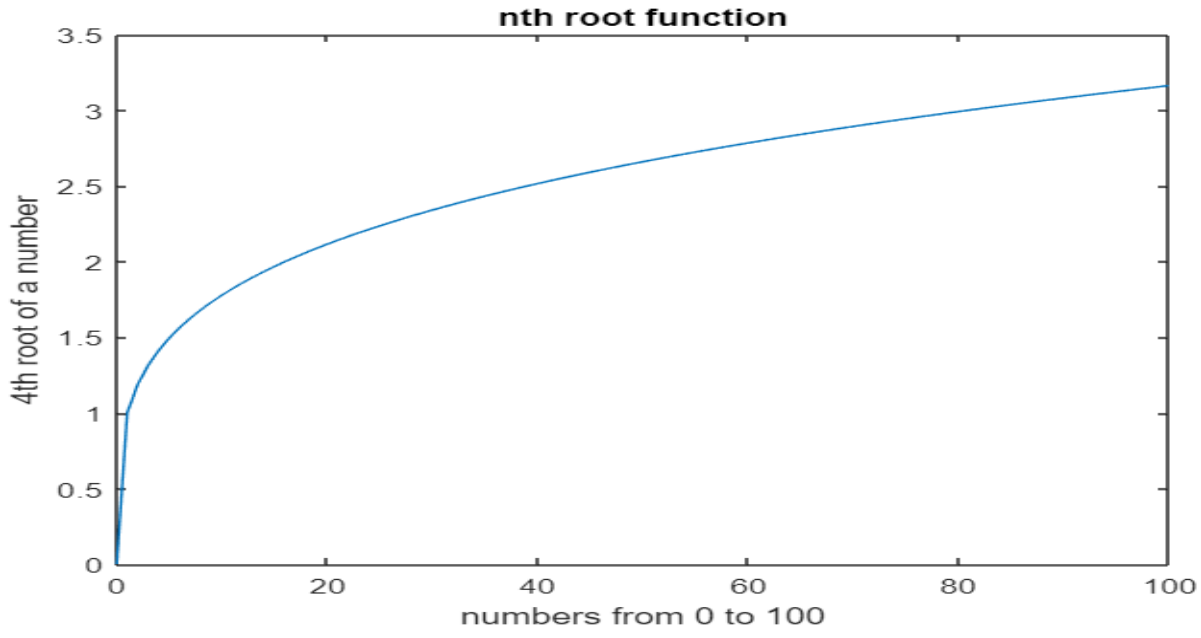
plot(X,Y);

title('nth root function');

xlabel('numbers from 0 to 100');

ylabel('4th root of a number');

Output>



Result: Rounding off number functions, trigonometric and logarithmic functions have been performed using MATLAB.

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 suitable curves)on Rectangular Plot.

Tools Used: MATLAB 7.0.

Theory:

Exp():

The exp function is an elementary of X. For complex numbers, it returns the complex exponential.

Subplot():

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.

Plot():

The plot function plots Y versus X. If X and Y are both matrices, then they must have equal size. The plot function plots columns of Y versus columns of X . If one of X or Y is a vector and the other is a matrix, then the matrix must have dimensions such that one of its dimensions equals the vector length.

Title():

The title(txt) function adds the specified title to the chart.

MATLAB Programs and Output:

Input>
```
n=0:100
xn=[(-1).^(n+1)]./(2*n-1);
y=xn;sum(y);
display(y);
plot(y);
```
Output>
```
y =

 Columns 1 through 6:

   1.0000e+00   1.0000e+00  -3.3333e-01   2.0000e-01  -1.4286e-01   1.1111e-01

 Columns 7 through 12:

  -9.0909e-02   7.6923e-02  -6.6667e-02   5.8824e-02  -5.2632e-02   4.7619e-02

 Columns 13 through 18:
```

-4.3478e-02 4.0000e-02 -3.7037e-02 3.4483e-02 -3.2258e-02 3.0303e-02

Columns 19 through 24:

-2.8571e-02 2.7027e-02 -2.5641e-02 2.4390e-02 -2.3256e-02 2.2222e-02

Columns 25 through 30:

-2.1277e-02 2.0408e-02 -1.9608e-02 1.8868e-02 -1.8182e-02 1.7544e-02

Columns 31 through 36:

-1.6949e-02 1.6393e-02 -1.5873e-02 1.5385e-02 -1.4925e-02 1.4493e-02

Columns 37 through 42:

-1.4085e-02 1.3699e-02 -1.3333e-02 1.2987e-02 -1.2658e-02 1.2346e-02

Columns 43 through 48:

-1.2048e-02 1.1765e-02 -1.1494e-02 1.1236e-02 -1.0989e-02 1.0753e-02

Columns 49 through 54:

-1.0526e-02 1.0309e-02 -1.0101e-02 9.9010e-03 -9.7087e-03 9.5238e-03

Columns 55 through 60:

-9.3458e-03 9.1743e-03 -9.0090e-03 8.8496e-03 -8.6957e-03 8.5470e-03

Columns 61 through 66:

-8.4034e-03 8.2645e-03 -8.1301e-03 8.0000e-03 -7.8740e-03 7.7519e-03

Columns 67 through 72:

-7.6336e-03 7.5188e-03 -7.4074e-03 7.2993e-03 -7.1942e-03 7.0922e-03

Columns 73 through 78:

-6.9930e-03 6.8966e-03 -6.8027e-03 6.7114e-03 -6.6225e-03 6.5359e-03

Columns 79 through 84:

-6.4516e-03 6.3694e-03 -6.2893e-03 6.2112e-03 -6.1350e-03 6.0606e-03

Columns 85 through 90:

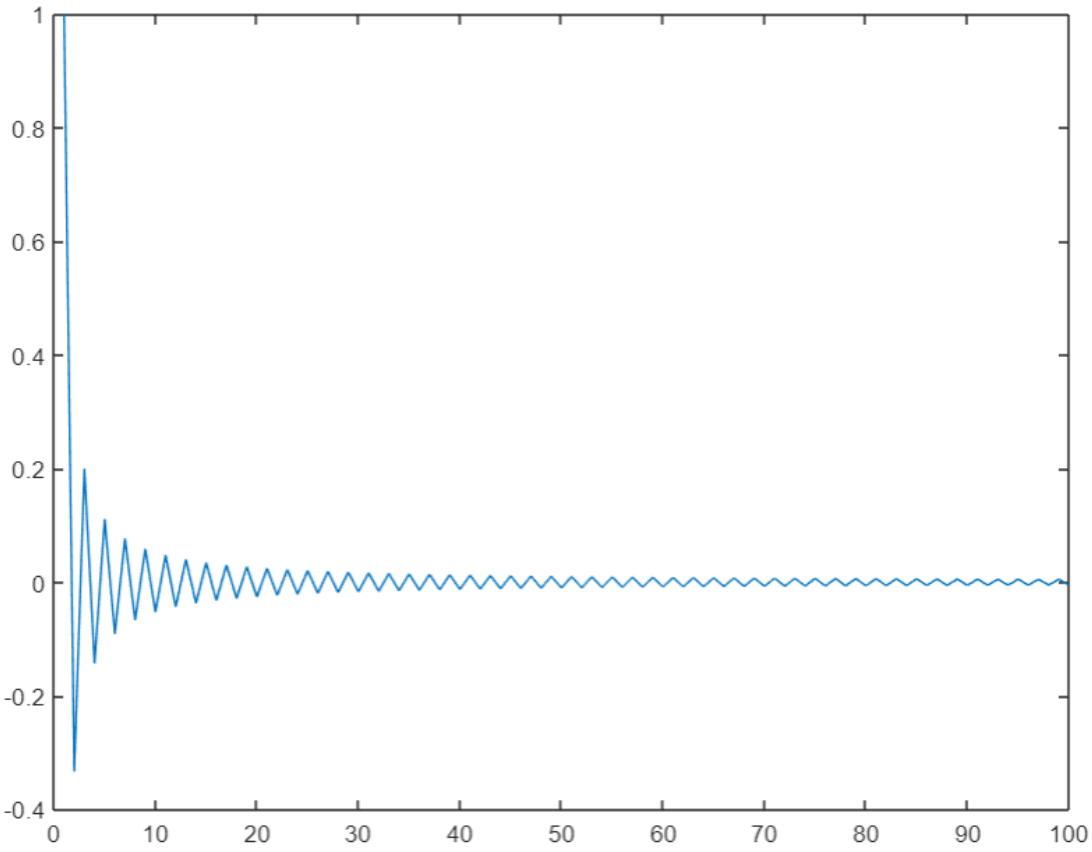
-5.9880e-03 5.9172e-03 -5.8480e-03 5.7803e-03 -5.7143e-03 5.6497e-03

Columns 91 through 96:

-5.5866e-03 5.5249e-03 -5.4645e-03 5.4054e-03 -5.3476e-03 5.2910e-03

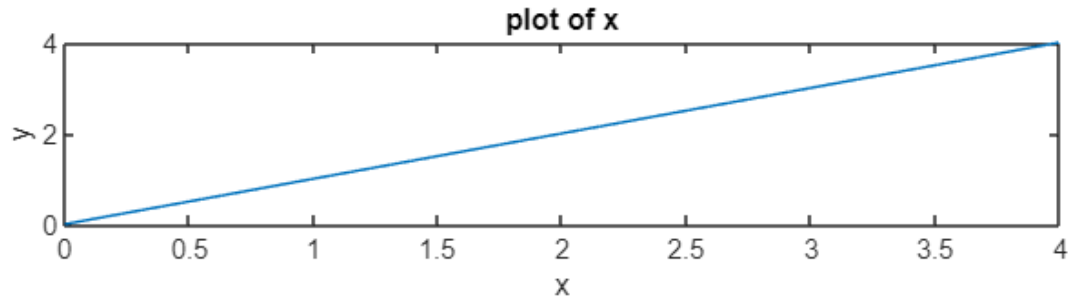
Columns 97 through 101:

-5.2356e-03 5.1813e-03 -5.1282e-03 5.0761e-03 -5.0251e-03

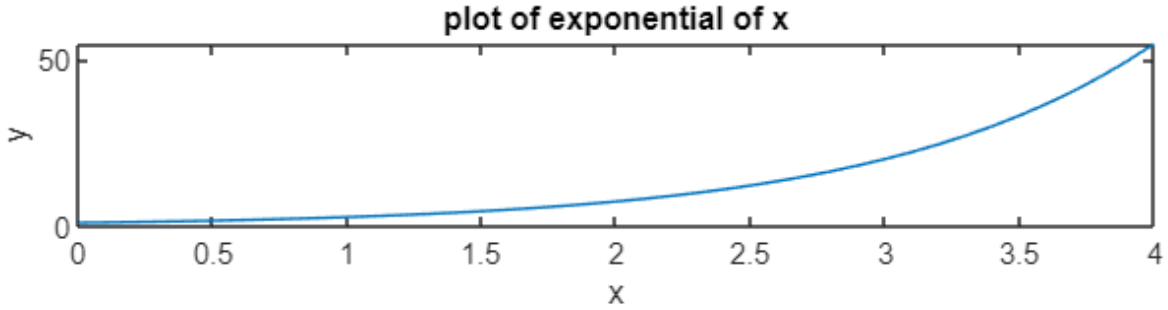


Input> x=0:0.1:4;
y=x;
subplot(3,1,1);
plot(x,y);
title('plot of x');
xlabel('x');
ylabel('y');

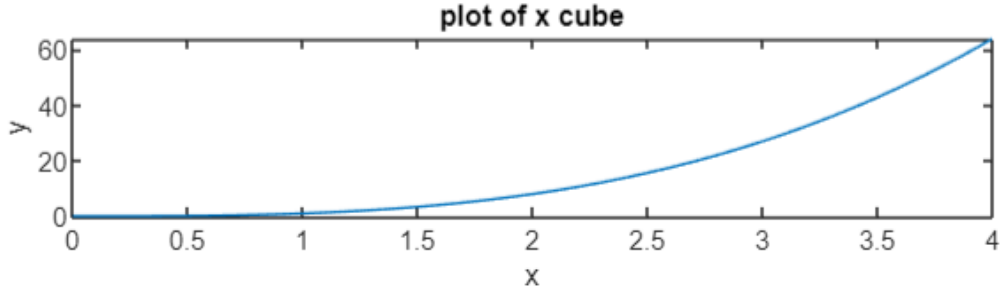
Output>



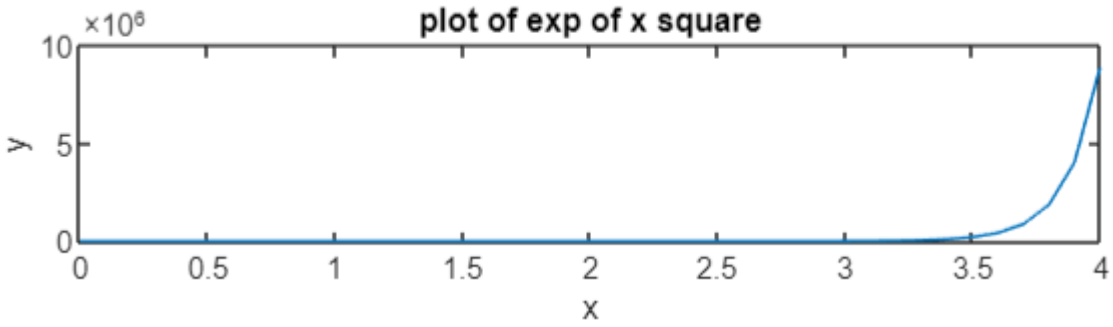
Input> x=0:0.1:4;
y=exp(x);
subplot(3,1,2);
plot(x,y);
title('plot of exponential of x');
xlabel('x');
ylabel('y');
Output>



Input>x=0:0.1:4;
y=x.^3;
subplot(3,1,3);
plot(x,y);
title('plot of x cube');
xlabel('x');
ylabel('y');
Output>



Input> x=0:0.1:4;
y=exp(x.^2);
subplot(3,1,3);
plot(x,y);
title('plot of exponential of x square');
xlabel('x');
ylabel('y');
Output>



Input> n=0:100;
y=sum(x);
xn=[(-1).^(n+1)]./(2*n-1);
y=xn;
sum(y);
display(y);
plot(y);
y=sum(xn);
display(y);
Output>y =

Columns 1 through 6:

1.0000e+00 1.0000e+00 -3.3333e-01 2.0000e-01 -1.4286e-01 1.1111e-01

Columns 7 through 12:

-9.0909e-02 7.6923e-02 -6.6667e-02 5.8824e-02 -5.2632e-02 4.7619e-02

Columns 13 through 18:

-4.3478e-02 4.0000e-02 -3.7037e-02 3.4483e-02 -3.2258e-02 3.0303e-02

Columns 19 through 24:

-2.8571e-02 2.7027e-02 -2.5641e-02 2.4390e-02 -2.3256e-02 2.2222e-02

Columns 25 through 30:

-2.1277e-02 2.0408e-02 -1.9608e-02 1.8868e-02 -1.8182e-02 1.7544e-02

Columns 31 through 36:

-1.6949e-02 1.6393e-02 -1.5873e-02 1.5385e-02 -1.4925e-02 1.4493e-02

Columns 37 through 42:

-1.4085e-02 1.3699e-02 -1.3333e-02 1.2987e-02 -1.2658e-02 1.2346e-02

Columns 43 through 48:

-1.2048e-02 1.1765e-02 -1.1494e-02 1.1236e-02 -1.0989e-02 1.0753e-02

Columns 49 through 54:

-1.0526e-02 1.0309e-02 -1.0101e-02 9.9010e-03 -9.7087e-03 9.5238e-03

Columns 55 through 60:

-9.3458e-03 9.1743e-03 -9.0090e-03 8.8496e-03 -8.6957e-03 8.5470e-03

Columns 61 through 66:

-8.4034e-03 8.2645e-03 -8.1301e-03 8.0000e-03 -7.8740e-03 7.7519e-03

Columns 67 through 72:

-7.6336e-03 7.5188e-03 -7.4074e-03 7.2993e-03 -7.1942e-03 7.0922e-03

Columns 73 through 78:

-6.9930e-03 6.8966e-03 -6.8027e-03 6.7114e-03 -6.6225e-03 6.5359e-03

Columns 79 through 84:

-6.4516e-03 6.3694e-03 -6.2893e-03 6.2112e-03 -6.1350e-03 6.0606e-03

Columns 85 through 90:

-5.9880e-03 5.9172e-03 -5.8480e-03 5.7803e-03 -5.7143e-03 5.6497e-03

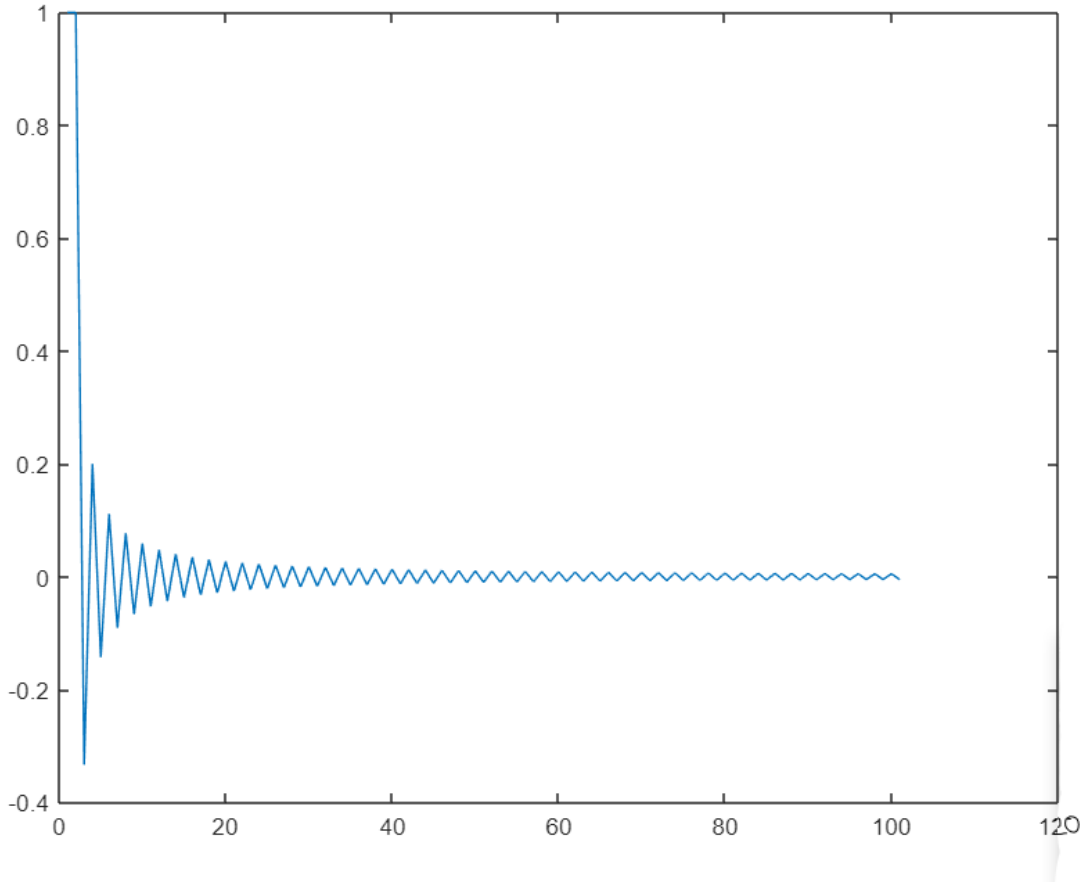
Columns 91 through 96:

-5.5866e-03 5.5249e-03 -5.4645e-03 5.4054e-03 -5.3476e-03 5.2910e-03

Columns 97 through 101:

-5.2356e-03 5.1813e-03 -5.1282e-03 5.0761e-03 -5.0251e-03

y = 1.7829



Result: Creation of 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, exp(x), exp(x2), over the interval 0<x<4 on Rectangular Plot has been done successfully.

Experiment 6

Aim: Generating a sinusoidal signal of a given frequency (say 100Hz) and plotting with graphical enhancements- Titling, Labelling, adding text, adding legends, adding new plots to existing plot, Plotting as multiple and subplot.

Tools Used: MATLAB 7.0

Theory:

- Hold on is used to plot more than one function in the same graph. 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.
- Labeling: Each axes graphics object can have one label for the x-, y-, and z-axis.The label appears beneath its respective axis in a two-dimensional plot and to the side or beneath the axis in a three-dimensional plot. For example: xlabel(' string ') labels the x-axis of the current axes.
- Grid on command is used to draw the grids in the function.
- Scatter(): scatter(x,y) creates a scatter plot with circles at the locations specified by the vectors x and y. This type of graph is also known as a bubble plot.
- Axis(): The axis command allows you to set the axis scales. You can provide minimum and maximum values of x and y axis using the axis command in the following way:Axis([xmin,xmax,ymin,ymax]);
- Legend:legend(label1,...,labelN) sets the legend labels. Specify the labels as a list of character vectors or strings, such as legend('Jan','Feb','Mar').
- MATLAB offers 8 basic colour options:

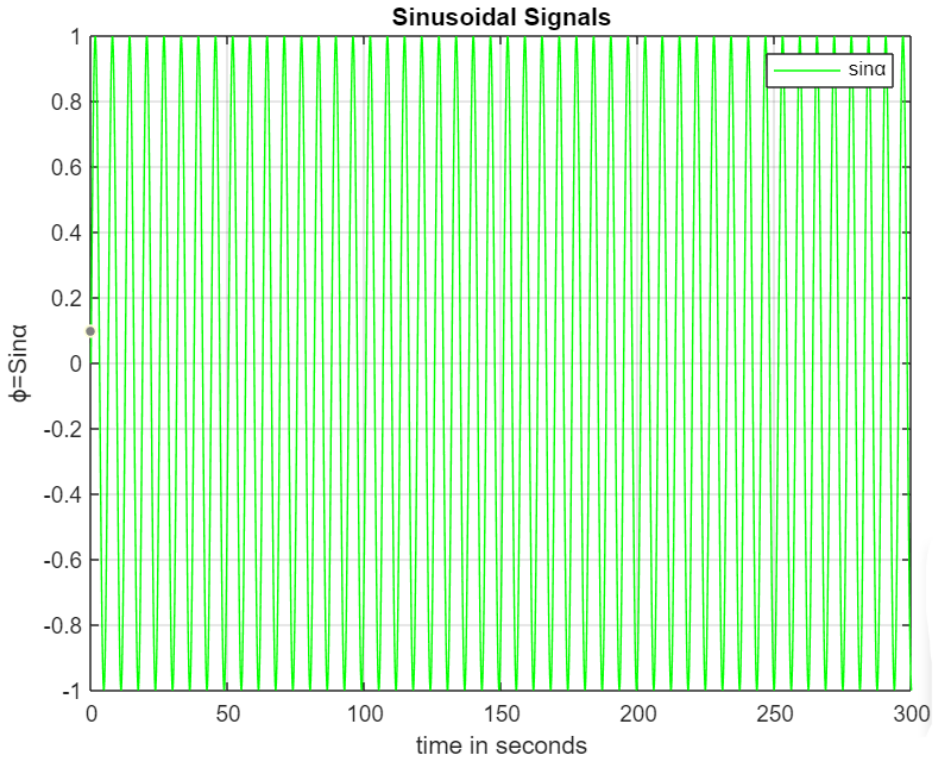
Color	Code
White	**w**
Black	**k**
Blue	**b**
Red	**r**
Cyan	**c**
Green	**g**
Magenta	**m**
Yellow	**y**

MATLAB Programs and Output:

(A) %code 1%
Input> x=0:0.100:100*pi;
y=sin(x);
plot(x,y,'g');
title('sinusoidal signal');

```
grid on;
xlabel('time in seconds');
ylabel('\phi=sin{\alpha}');
legend('sin\alpha');
axis([0,300,-1,1]);
```

Output>

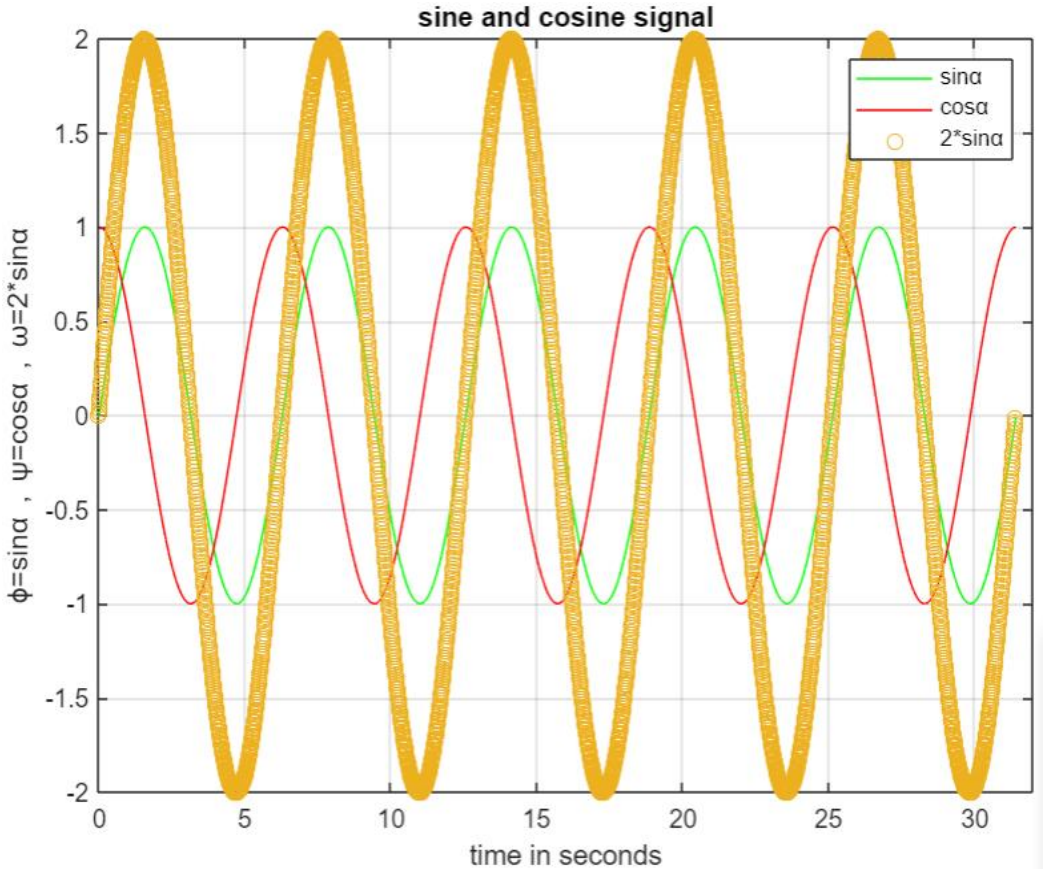


(B) %code 2%

Input>

```
x=0:0.001:100*pi;
y=sin(x);
plot(x,y,'g');
grid on;
hold on;
y1=cos(x);
plot(x,y1,'r');
hold on;
a=2;
y2=a*sin(x);
scatter(x,y2);
hold off;
axis([0,32,-2,2]);title('sine and cosine signals');
xlabel('time in seconds');
ylabel('\phi=sin\alpha,\psi=cos\alpha,\omega=2*sin\alpha');
legend('sin\alpha','cos\alpha','2*sin\alpha');
```

Output>



Result: The generation of the sinusoidal signal of a given frequency has been done successfully.

Tutorial Sheet -1

Q1)Solve for x

Ax=Y

A= 9 -36 30

-36 192 -180

30 -180 180

Y= 1

0

0

Ans)

Input> A=[9,-36,30;-36,192,-180;30,-180,180];

display(A);

Y=[1;0;0];

display(Y);

B=inv(A);

display(B);

X=B*Y;

display(X);

Output >

A =

9 -36 30

-36 192 -180

30 -180 180

Y =

1

0

0

B =

1.0000 0.5000 0.3333

0.5000 0.3333 0.2500

0.3333 0.2500 0.2000

X =

1.0000

0.5000

0.3333

Q2)What is A^n

A= 0.99 0.01

-0,01 1.01

Ans)

Input> A=[0.99,0.01;-0.01,1.01];

display(A);

n=4;

n=5;

An=nthroot(A,n);

display(An);

Output >

A =

0.990000 0.010000

-0.010000 1.010000

An =

0.9980 0.3981

-0.3981 1.0020

Q3) Compute the following

A) $(-3)^{1/2}$

B) Sin(∏/3)

C) $(-2)^{-4/3}$

D) $Sin(1^0)$

Ans)

(A)

Input> A=(-3)^(1/4);

display(A);

Output>

A = 0.9306 + 0.9306i

(B)

Input>

x=pi/3;

B=sin(x);

display(B);
Output>
B = 0.8660
(C)
Input>
C=(-2)^(-4/3);
display(C);
Output>
C = -0.1984 + 0.3437i
(D)
Input>
y=pi/180;
D=sin(y);
display(D);
Output>
D = 0.017452

Q4)Write the matlab code for converting temperature given in celsius to Fahrenheit

Ans)
Input> C=37;
x=9/5;

D=c*x;

D=C*x;

display(C);

display(D);

F=D+32;

display(F);

Output>

D = 66.600

C = 37

F = 98.600

Q5)Solve the following equation
$X=1/(1+X^2)^{1/2}$

Ans)
Input> a=-(1+sqrt(5))/2;

```
b=-(1-sqrt(5))/2;
display(a);
display(b);
Y=sqrt(1+a);
display(Y);
X=1/Y;
display(X);
Z=sqrt(1+b);
display(Z);
X1=1/Z;
display(X1);
```

Output>

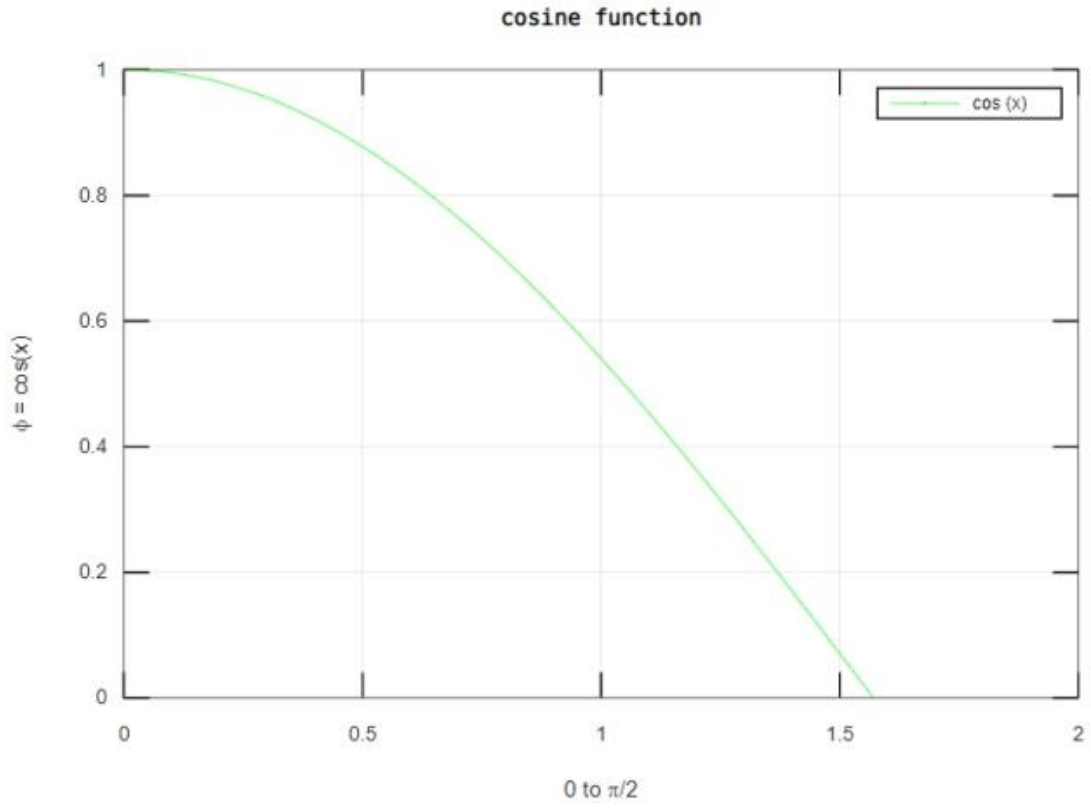
```
a = -1.6180
b = 0.6180
Y =       0 + 0.7862i
X =       0 - 1.2720i
Z = 1.2720
X1 = 0.7862
```

Q6)Find the solution
X=cos x for the interval (0,pi/2)

Ans)

```
Input> x=0:0.01:pi/2;
y=cos(x);
plot(x,y,'g');
title('cosine function');
xlabel('0 to \pi/2');
ylabel('\phi = cos(x)');
grid on;
legend('cos (x)');
```

Output>



Q7) X^8 . Find a real 2 by 2 matrix X so that X^8 =-1

Ans)

Input> X=[1,0;0,1];

display(X);

X=-X;

display(X);

R=X^(1/8);

display(R);

Output>

X =

-1 0

0 -1

X =

1 0

0 1

R =

0.9239 + 0.3827i 0 + 0i

0 + 0i 0.9239 + 0.3827i

Q8) If N= Magic(4) then compute $(N^2+N)/2$

Ans)

Input>N=magic(4);
display(N);

M=((N^2)+N)/2;

M=((N^3)+N)/2;

display(M);

Output>

N =

16 2 3 13

5 11 10 8

9 7 6 12

4 14 15 1

M =

5221.0 4654.0 4694.5 5099.5

4775.5 5018.5 4978.0 4897.0

4937.5 4856.5 4816.0 5059.0

4735.0 5140.0 5180.5 4613.5

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

=(0.45T+900) for T>100

Exercise: Testing the scripts written using

(C) T=5 and h=-5

(D) T=110 and h=949.5

Tools Used: MATLAB 7.0/ Octave Online

Theory: When using if, else if, else statements there are few points to keep in mind:

1. An if can have zero or one else's and it must come after any else ifs.
2. An if can have zero to many else if's and they must come before the else.
3. Once an else if succeeds, none of the remaining else if's or else's will be tested.

MATLAB Programs and Output:

(1) **Input**>T=input('enter the value of T for the function h(T)');
if(T>0 && T<100)
h=T-10;
display('value of h');
display(h);
elseif(T>100)
h=0.45*T;
h=h+900;
display('value of h');
display(h);
elseif(T<0)
T=0;
else
display('error');
end;
Output>
enter the value of T for the function h(T)> 12
value of h
h = 2

(2) **Input**>T=input('enter the value of T for the function h(t)');
if(T>0 && T<100)
display('value of h');
h=T-10;
elseif(T>100)
display('value of h');
h=h*0.45;

```
h=t*0.45;
h=T*0.45;
h=h+900;
elseif(T<0)
T=0;
else
display('error');
end;
```

Output>

```
enter the value of T for the function h(t)> 0
error
```

Results and Conclusion: Evaluation of function h(T) using if else statements has been done successfully.

Experiment 9

Aim: Generating a square wave from sum of sine waves of certain amplitude and frequencies.

The Gibbs phenomenon involves both the fact that Fourier sums overshoot at a jump discontinuity, and that this overshoot does not die out as the frequency increases

$$\sin(x) + \frac{1}{3}\sin(3x) + \frac{1}{5}\sin(5x) \ldots$$

Tools Used: MATLAB 7.0

Theory:

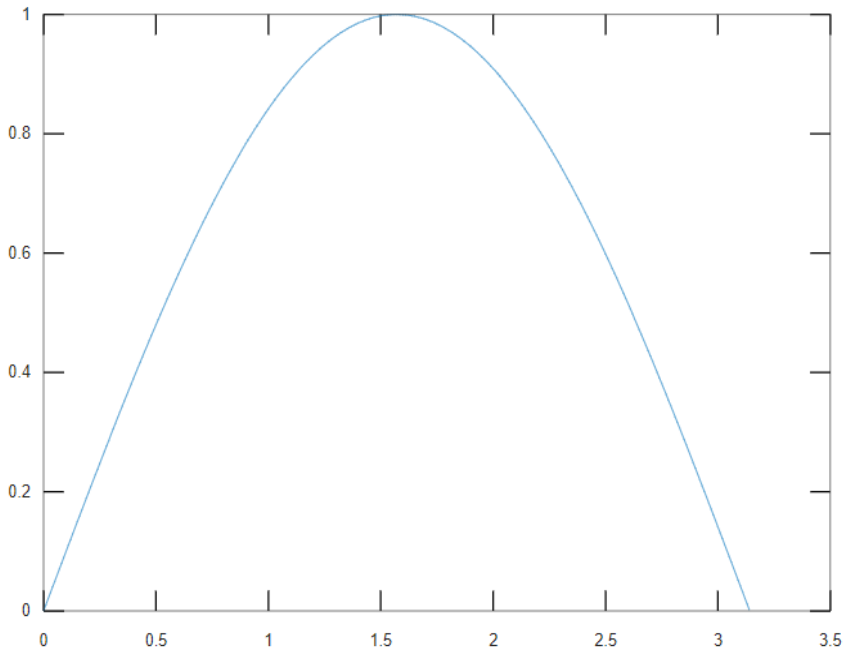
Procedure:

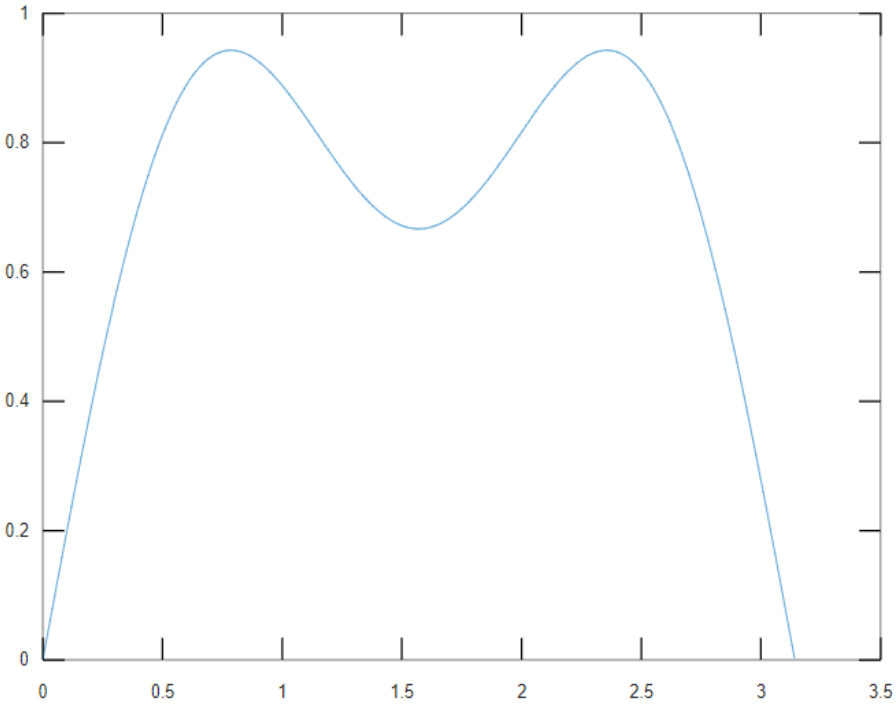
1. The Fourier series expansion for a square-wave is made up of a sum of odd harmonics.
2. We start by forming a time vector running from 0 to 10 in steps of 0.1, and take the sine of all the points.
3. Now use the first, third, fifth, seventh and ninth harmonics.
4. For a finale, we will go from the fundamental to the 19th harmonic, creating % vectors of successively more harmonics and saving all intermediate steps as % the rows of a matrix.
5. These vectors are plotted on the same figure to show evolution of the square wave. Note Gibb's effect.

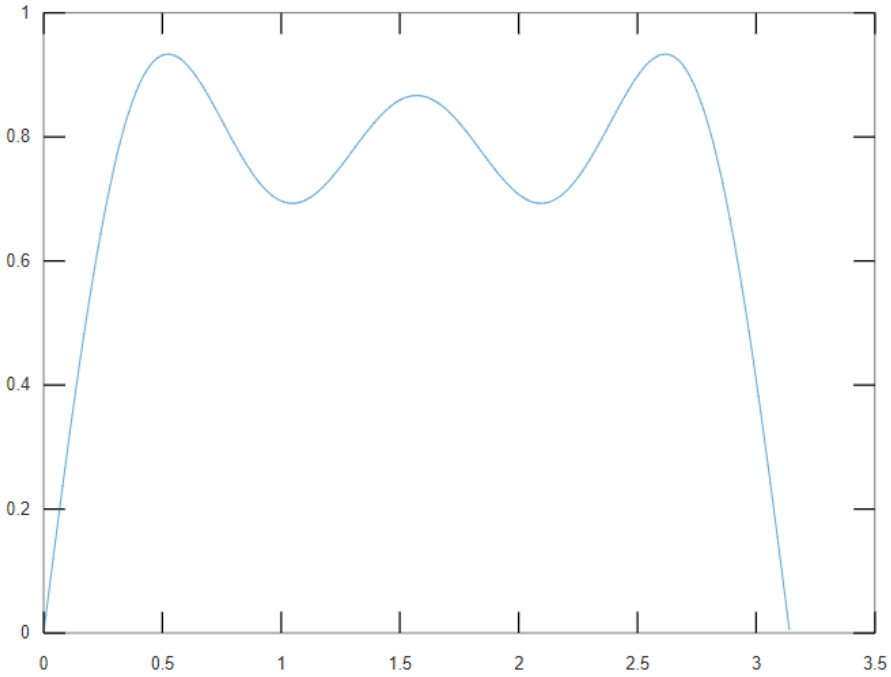
MATLAB Programs and Output:

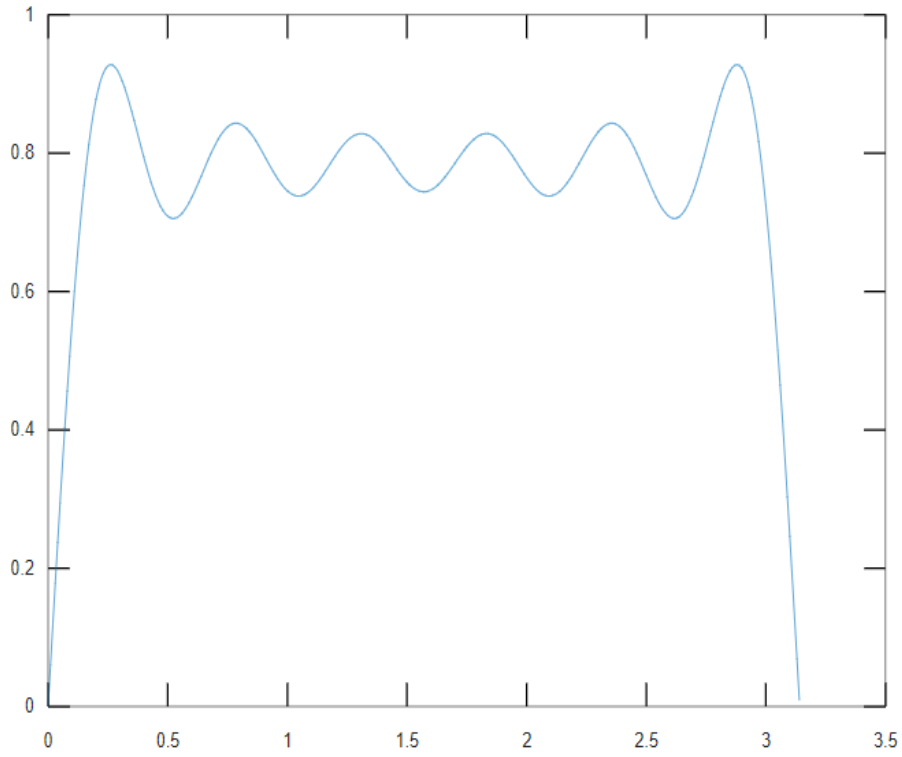
(1) %By Conventional Method%

Input>x=0:0.01:pi;
y=sin(x);
plot(x,y);
y=y+sin(3*x)./3;
plot(x,y);
y=y+sin(5*x)./5;
plot(x,y);
y=y+sin(7*x)./7+sin(9*x)./9+sin(11*x)./11;
plot(x,y);
y=y+sin(13*x)./13+sin(15*x)./15+sin(17*x)./17;
plot(x,y);
hold on;
y2=sin(x);
plot(x,y2,'g');
hold off;
grid on;
Output>

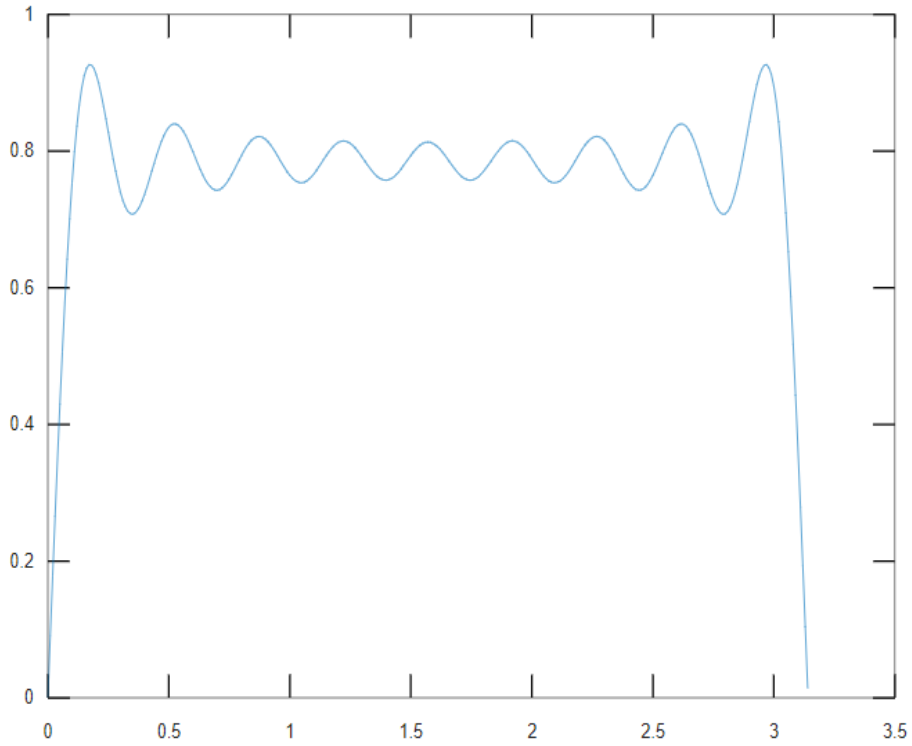

1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5


1
0.8
0.6
0.4
0.2
0
0
0.5
1
1.5
2
2.5
3
3.5

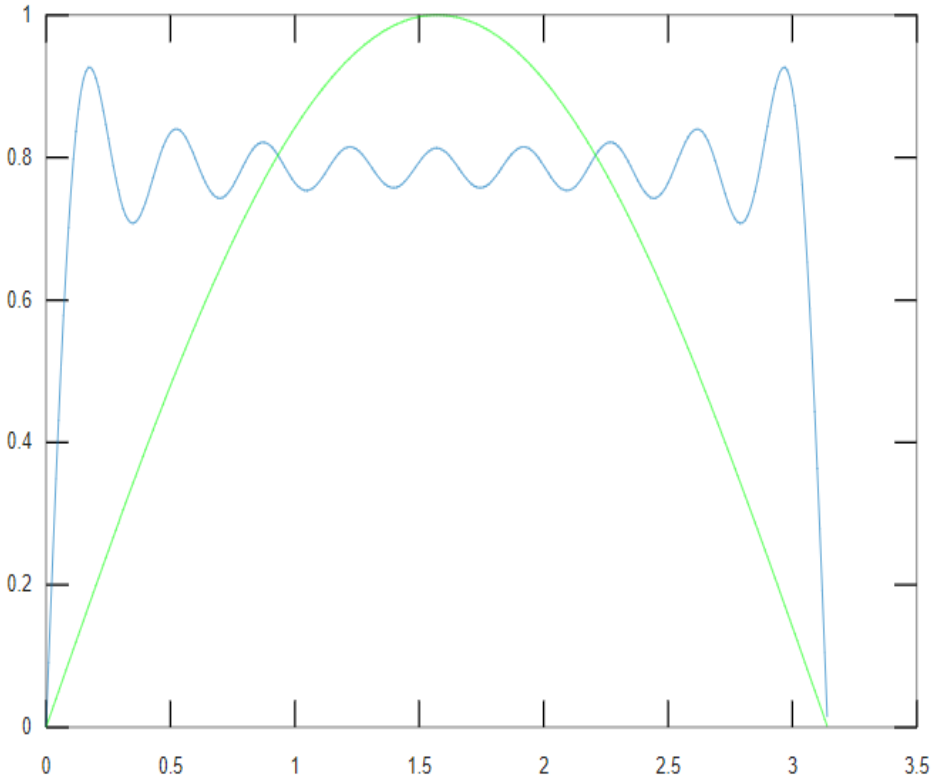


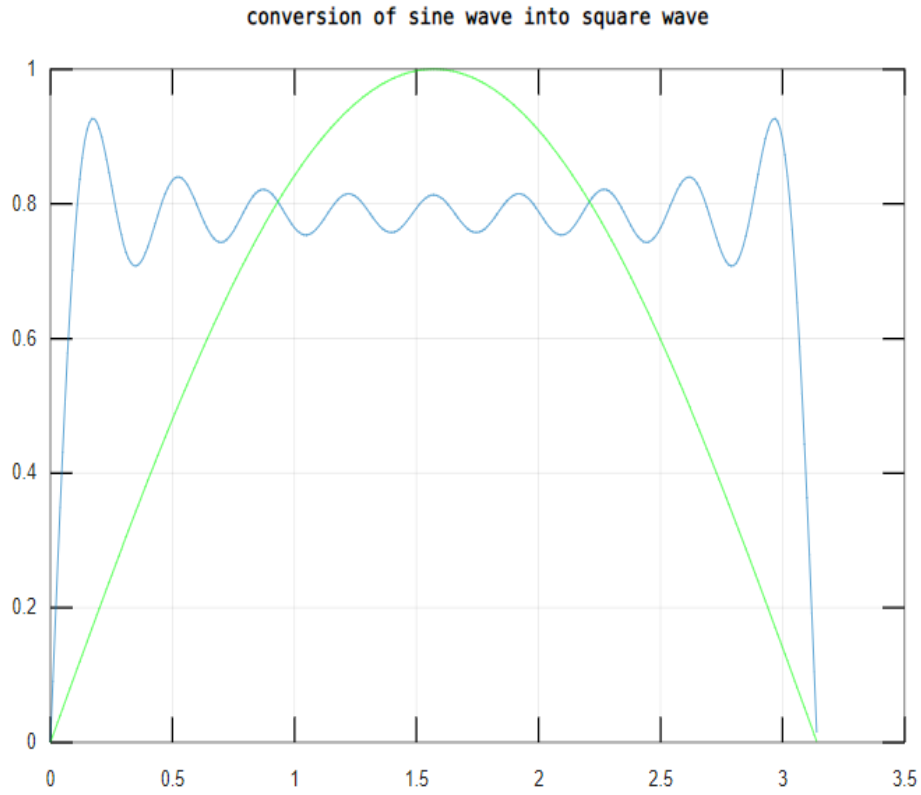


Shaina Mehta



conversion of sine wave into square wave

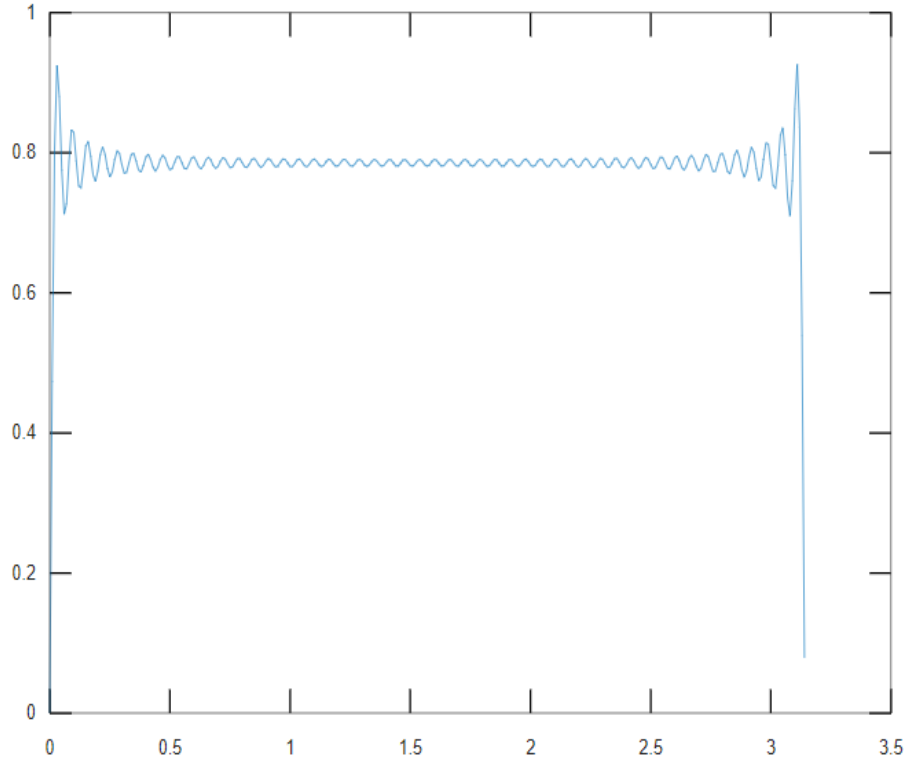


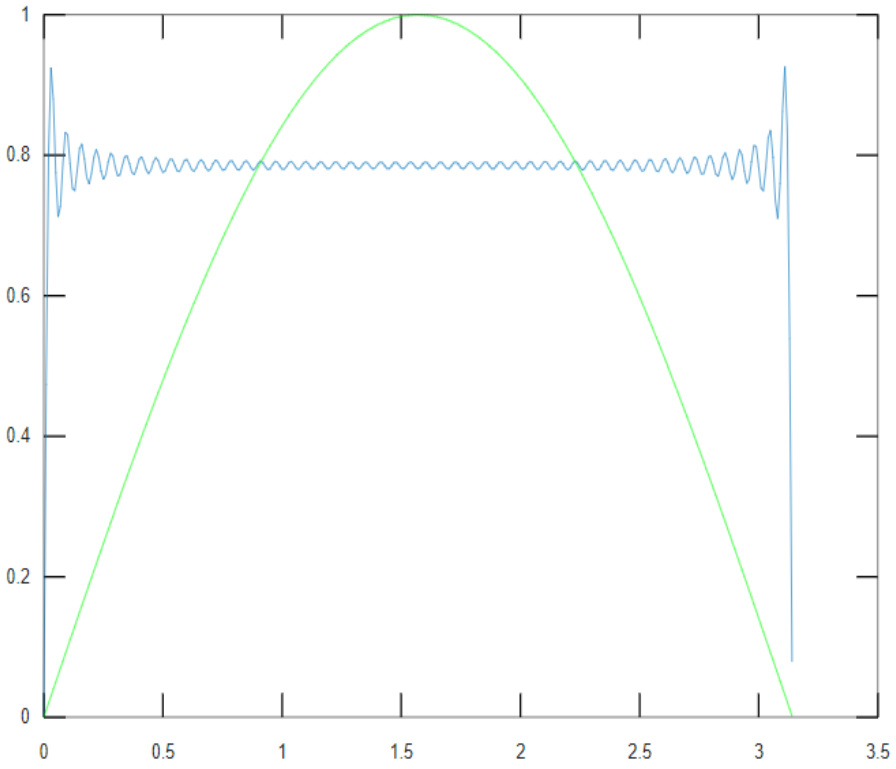


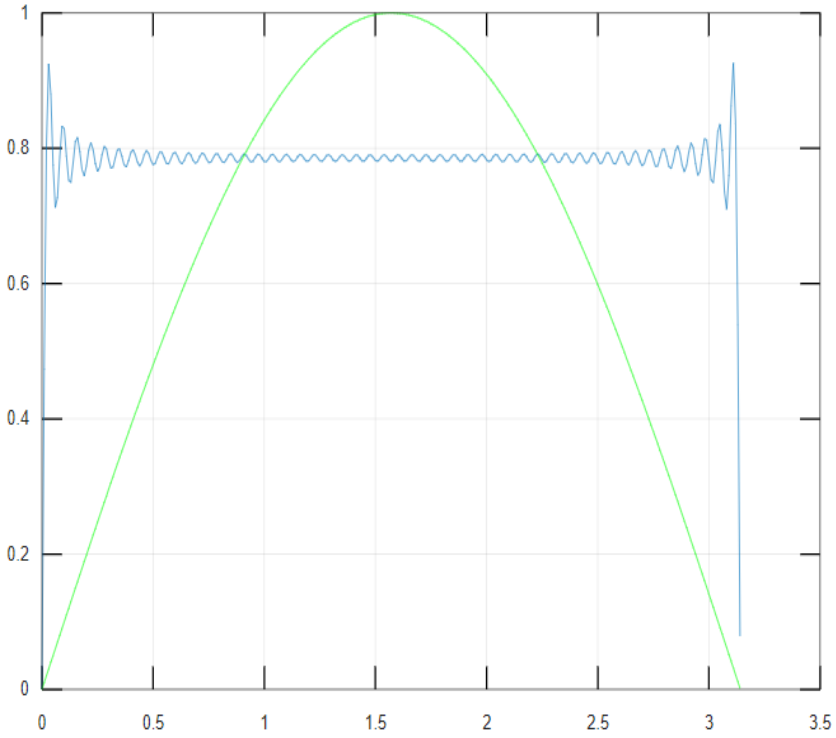
(2) %Using For Loop%

Input> x=0:0.01:pi;
y=zeros(1,length(x));
for n=1:2:99
y=y+sin(n*x)./n;
end;
plot(x,y);
hold on;
y2=sin(x);
plot(x,y2,'g');
hold off;
grid on;

Output>







Results and Conclusion: A square wave from sum of sine waves of certain amplitude and frequencies has been generated successfully.

Experiment 7

Aim: Solving First Order Ordinary Differential Equation using Built-in Functions.

Tools Used: MATLAB 7.0 / Octave.

Theory: Symbolic functions represent math functions. Symbolic functions are used for differentiation, integration, solving ODEs, and other math operations.

- **syms**: Creates symbolic variables and functions.
  - Creates symbolic functions with one and two arguments (syms f(x,y)).
  - Creates a symbolic function and can specify its formula by using a symbolic matrix (syms f(x))
- Using ode23 and ode45,
  [t,y] = ode23(odefun,tspan,y0,options)
  [t,y] = ode45(odefun,tspan,y0,options)

Programs and Output:

Input>
```
syms f(x,y);
f(x,y)=x^2*y;
d=diff(f(x,y),x);
display(d);
di=diff(d,y);
display(di);
syms y(t) a;
e=diff(y,t)==a*y^2;
c=y(0)==5;
sol=dsolve(e,c);
display(sol);
syms g(x);
g(x)=[x,x^2;x^3,x^4];
display(g(x));
df=diff(g,x);
display(df);
df(3);
display(df(3));
```
Output>

d = (sym) 2·x·y;

di = (sym) 2·x;

sol = (sym)

$$y(t) = \frac{-1}{a \cdot t - 1/5}$$

(sym 2×2 matrix)

$$\begin{bmatrix} & 2 \\ x & x \end{bmatrix}$$

$$\begin{bmatrix} & \\ 3 & 4 \\ x & x \end{bmatrix}$$

df(x) = (symfun)

$$\begin{bmatrix} 1 & 2\cdot x \\ & \\ 2 & 3 \\ 3\cdot x & 4\cdot x \end{bmatrix}$$

(sym 2×2 matrix)

$$\begin{bmatrix} 1 & 6 \\ & \\ 27 & 108 \end{bmatrix}$$

Results and Conclusion: The ordinary and the partial differential equation has been solved successfully in MATLAB.

Experiment 10

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

Tools Used: MATLAB 7.0 / Octave

Theory:

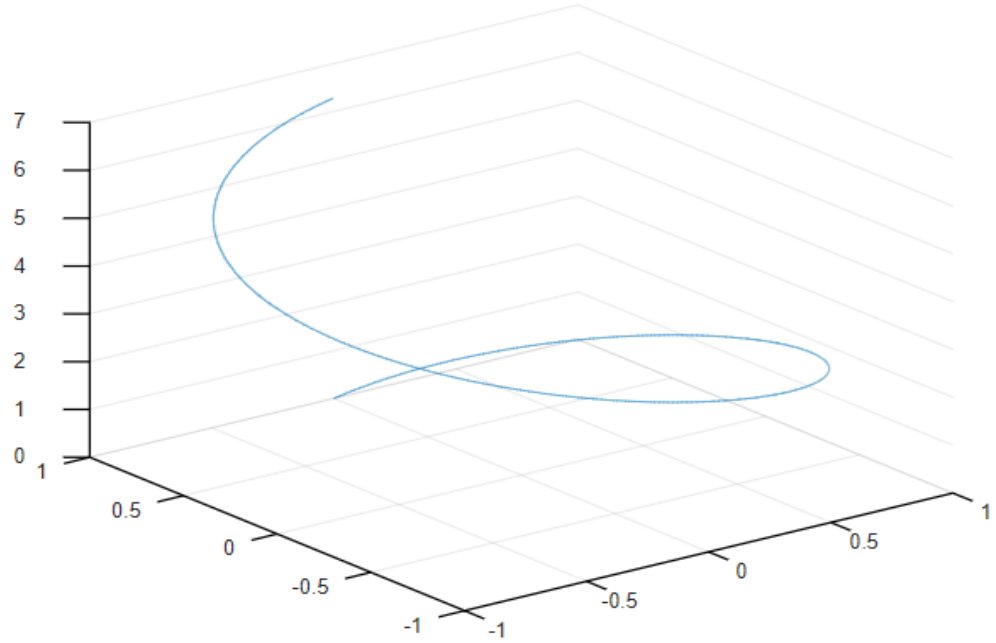
- Parameterized curve: A parameterized curve is a vector representation of a curve that lies in 2 or 3-dimensional space.
- Bar(): Bar graph
  - bar(y): bar(y) creates a bar graph with one bar for each element in y. If y is an *m*-by-*n* matrix, then bar creates *m* groups of *n* bars.
  - bar(x,y): bar(x,y) draws the bars at the locations specified by x.
- 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 othe number of elements in X.
  - pie(x,labels): pie(x,labels) specifies options for labelling the pie slices. In this case, x must be numeric.
- Surfc():
  - 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.
  - surfc(X,Y,Z,C): It additionally specifies the surface color.
  - 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.
- Meshgrid():
  - [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.
  - [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).
  - [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)-bylength(z).

- [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).

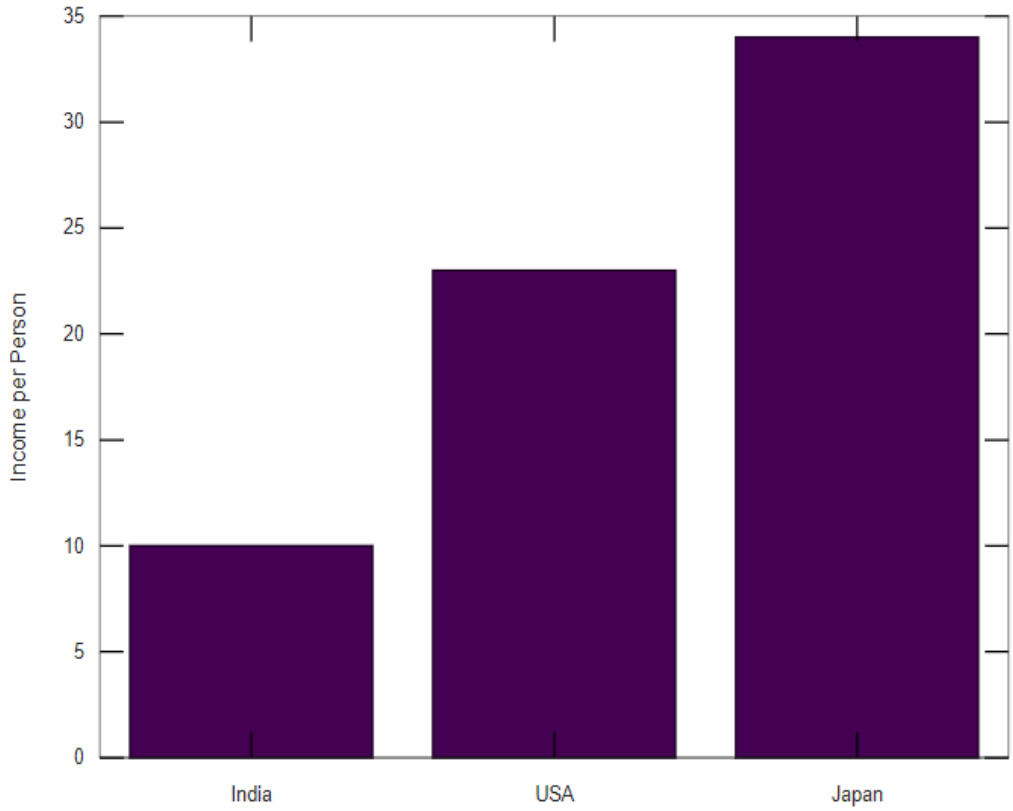
- Polyshape(): The polyshape function creates a polygon defined by 2-D vertices, and returns a polyshape object with properties describing its vertices, solid regions, and holes. For example, pgon = polyshape([0 0 1 1],[1 0 0 1]) creates the solid square defined by the four points (0,1), (0,0), (1,0), and (1,1).

Programs and Output:

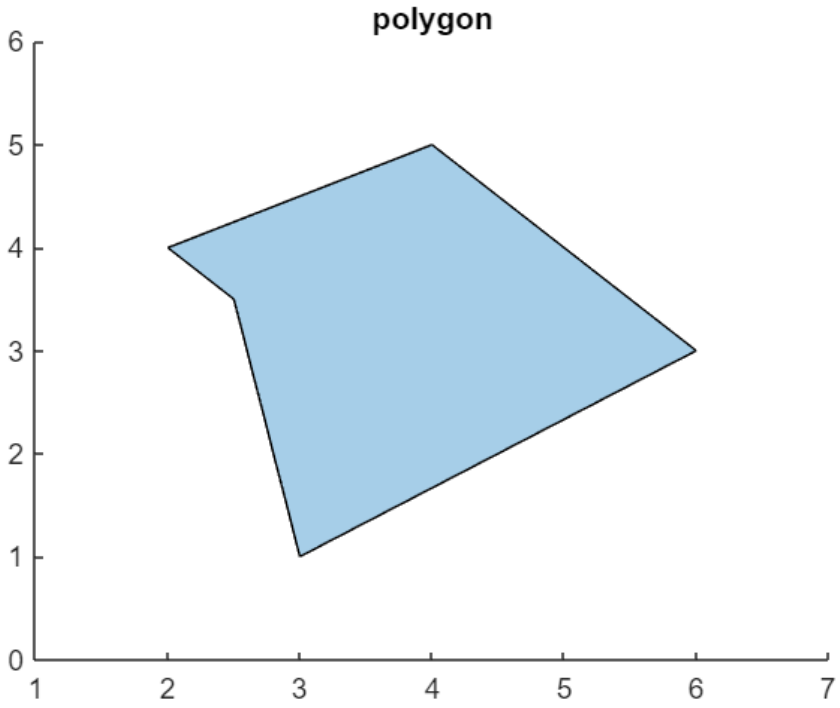
Input> syms;
Symbolic variables in current scope:
x=sin(t);
t=0:0.01:2*pi;
x=sin(t);
y=cos(t);
z=t;
plot3(x,y,z);
grid on;
Output>



Input> y=[10,23,34];
bar(y);
x=x=set(gca,"xticklabel",{"India","USA","Japan"});
ylabel("Income per Person");
Output>

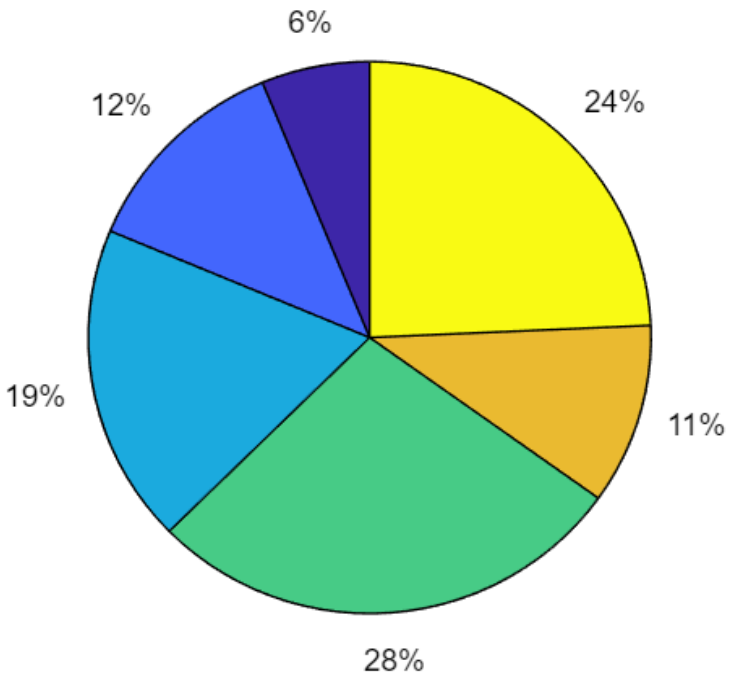


Input>p=polyshape([2 4 6 3 2.5],[4 5 3 1 3.5]);
plot(p);
title('polygon');
Output>



Input>Q=[20 40 60 89 34 78];
display(Q);

pie(Q);
Output>
Q =
20 40 60 89 34 78

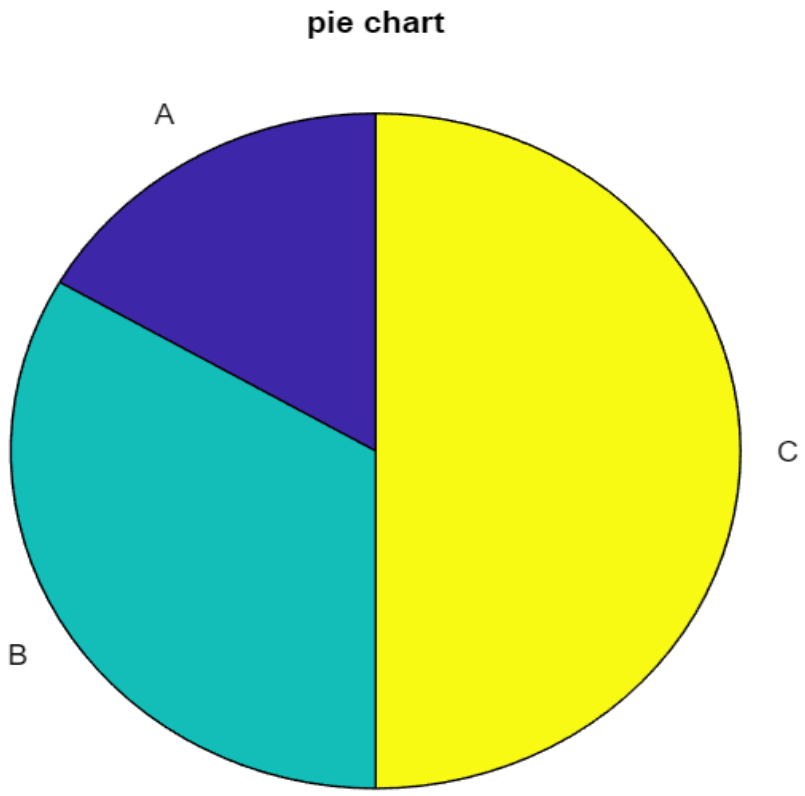


Input>A=[10,20,30];
display(A);
labels=["A","B","C"];
display(labels);
pie(A,labels);
title('pie chart');
Output>
A =

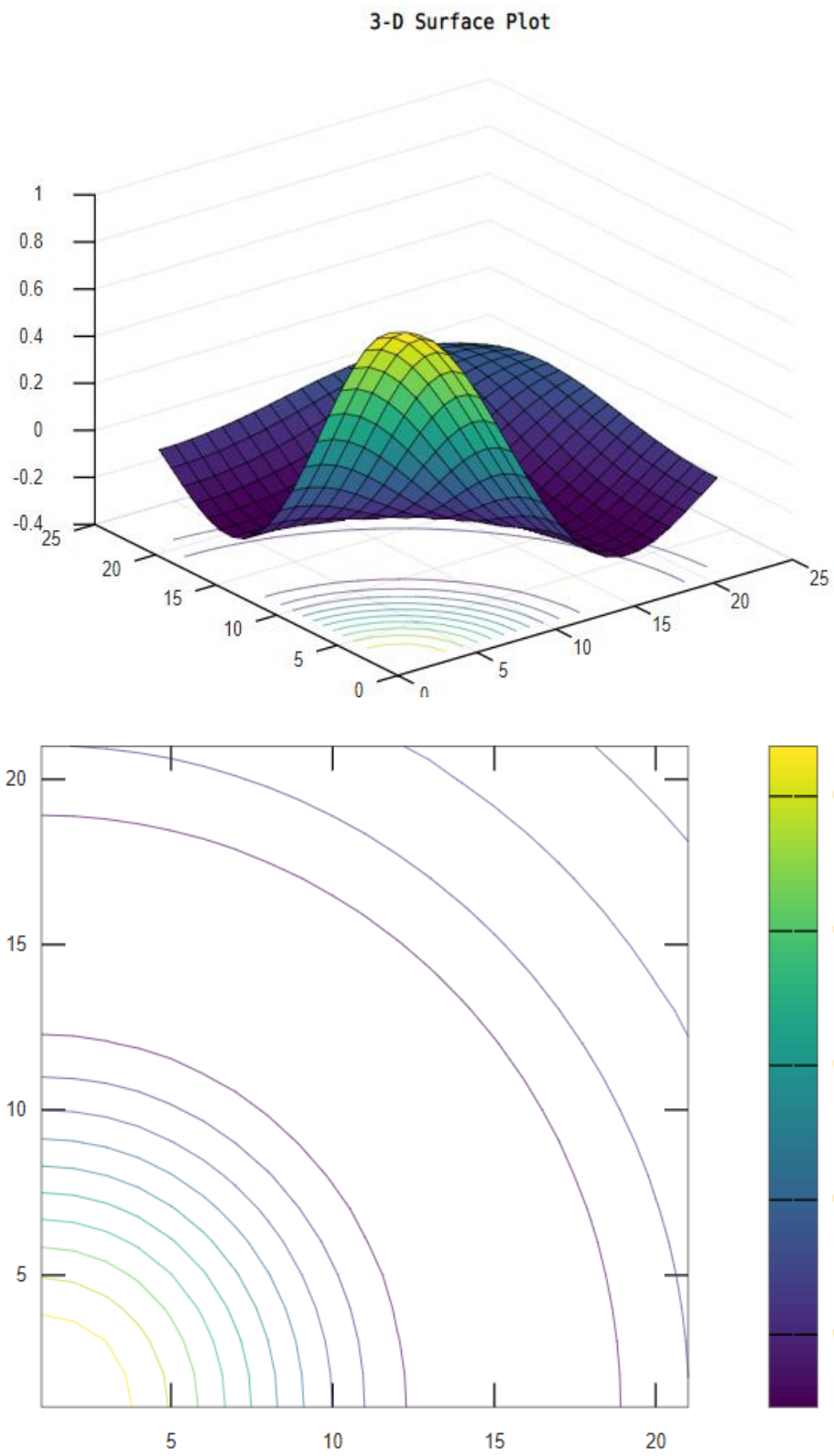
10 20 30
labels =

1×3 **string** array

"A" "B" "C"



Input>y1=0:pi/10:2*pi;
y2=0:pi/10:2*pi;
[X,Y]=meshgrid(y1,y2);
R=sqrt(X.^2+Y.^2);
Z=sin(R)./R;
surfc(Z);
title('3-D Surface Plot');
contour(Z);
colorbar;
Output>



Results and Conclusion: The basic 2D and 3D plots has been done successfully.