AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

AMITY UNIVERSITY

----------- UTTAR PRADESH-----------

BASIC SIMULATION LAB

LAB MANUAL

BASIC SIMULATION LAB

Student Learning Outcomes:

Outcome 1: (Scientific foundation) When faced with a technical problem the student should be able to use applied scientific knowledge

1A: to identify and implement relevant principles of mathematics and computer science.

1 B: to identify and implement relevant principles of physics and chemistry

1 C: to identify and implement relevant principles of engineering science

Outcome 2: (Experimentation) the ability to design experiments, conduct experiments, and analyze experimental data.
Outcome 3: (Tools) an ability to use the relevant tools necessary for engineering practice.
Outcome 4: (Technical design) the technical ability to design a prescribed engineering sub-system

Outcome 8: (Teamwork) the ability to function in teams.

Modules	Topics	REMARKS.
1	Creating a One-Dimensional Array (Row / Column Vector) ; 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.	PRACTICAL
2	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	PRACTICAL
3	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. Also, Generating a Random Sequence using rand() / randn() functions and plotting them.	PRACTICAL

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), $\log_{10}$(A), Square root of A, Real n^{th} root of A.	PRACTICAL
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, x^3, e^x, $\exp(x^2)$ over the interval $0 < x < 4$ (by choosing appropriate mesh values for x to obtain smooth curves), on A Rectangular Plot	PRACTICAL
6	Generating a Sinusoidal Signal of a given frequency (say, 100Hz) and Plotting with Graphical Enhancements - Titling, Labeling, Adding Text, Adding Legends, Adding New Plots to Existing Plot, Printing Text in Greek Letters, Plotting as Multiple and Subplot.	PRACTICAL
7	Solving First Order Ordinary Differential Equation using Built-in Functions.	PRACTICAL
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 $0 < T < 100$ $= (0.45\,T + 900)$ for $T > 100$. *Exercise : Testing the Scripts written using A). T = 5, h = -5 and B). T = 110, h =949.5*	PRACTICAL
9	**9**: Generating a Square Wave from sum of Sine Waves of certain Amplitude and Frequencies.	PRACTICAL
10	OPEN ENDED EXPERIMENT	PRACTICAL

Examination Scheme:

Components	**A**	**CT**	**S/V/Q**	**HA**	**EE**
Weightage (%)	5	10	8	7	70

CT: Class Test, HA: Home Assignment, S/V/Q: Seminar/Viva/Quiz, EE: End Semester Examination; A: Attendance

EXPERIMENT NO 1

AIM: Creating a One-Dimensional Array (Row / Column Vector) ; 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

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, Modeling, 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**.

Starting MATLAB on Windows platforms, start MATLAB by double-clicking the MATLAB shortcut icon on your Windows desktop.

Quitting MATLAB: To end your MATLAB session, select File > Exit MATLAB in the desktop, or type quit in the Command Window. You can run a script file named finish.m each time MATLAB quits.

Consider two matrices A and B. If A 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 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)th position, in the resulting matrix C, is the summation of the products of elements in ith row of first matrix with the corresponding element in the jth 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**.
In MATLAB, inverse of a matrix is calculated using the **inv** function. Inverse of a matrix A is given by inv(A).
The transpose operation switches the rows and columns in a matrix. It is represented by a single quote(').

The rank function provides an estimate of the number of linearly independent rows or columns of a full 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.

PROCEDURE:

To plot the graph of a function, you need to take the following steps:

1. Define **x**, by specifying the **range of values** for the variable **x**, for which the function is to be plotted.

2. Define the function, **y = f(x).**

3. Call the **plot** command, as **plot(x, y).**

SUGGESTED PROGRAM:

1)

```
A=[1 2 3 4];
display(A);
B=[5 6 7 8];
display(B);
```

```
ADD=A+B;
display(ADD);

SUBTRACT=A-B;
display(SUBTRACT);

MULTIPLY=2*A;
display(MULTIPLY);

DIVISION=B/2;
display(DIVISION);

EXP=exp(A);
display(EXP);
```

OUTPUT

```
A =

     1     2     3     4

B =

     5     6     7     8

ADD =

     6     8    10    12

SUBTRACT =

    -4    -4    -4    -4

MULTIPLY =

     2     4     6     8

DIVISION =

    2.5000    3.0000    3.5000    4.0000

EXP =

    2.7183    7.3891   20.0855   54.5982
```

b)

```
X=[1 20 3; 5 6 7; 8 9 10];
display(X);

RANK=rank(X);
display(RANK);

INVERSE=inv(X);
display(INVERSE);

TRANSPOSE=X';
display(TRANSPOSE);
```

<u>OUTPUT:</u>

```
X =
     1    20     3
     5     6     7
     8     9    10

RANK =

     3

INVERSE =

   -0.0278   -1.6019    1.1296
    0.0556   -0.1296    0.0741
   -0.0278    1.3981   -0.8704

TRANSPOSE =

     1     5     8
    20     6     9
     3     7    10
```

EXPERIMENT NO 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

TOOL USED: MATLAB 7.0

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:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operations
- 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:

PROCEDURE:

Operator	Description
+	Addition or 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, 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, For nonscalar A and B, the number of columns of A must equal 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, 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 operator works on bits and performs 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 are 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)	Set 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	Set exclusive OR of two arrays
union	Set union of two arrays
unique	Unique values in array

SUGGESTED PROGRAM:

Concatenation and Indexing

```
A=ones(2,3)*6;
display(A)
B=ones(3,3);
display(B)
C=[A;B];
display(C);
```

OUTPUT

```
A =


     6     6     6
     6     6     6

B =


     1     1     1
     1     1     1
     1     1     1

C =


     6     6     6
     6     6     6
     1     1     1
     1     1     1
     1     1     1
```

Random Matrices and Sorting

```
A=magic(3);
display(A);
A1=rand(3);
display(A1);
A2=pascal(3);
display(A2);
B=sort(A);
display(B);
C=sort(A,'descend');
display(C);
D=sort(A,'ascend');
display(D);
E=sortrows(A,1);
display(E);
```

OUTPUT`

```
A =


     8     1     6
     3     5     7
     4     9     2

A1 =


    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575

A2 =


     1     1     1
     1     2     3
     1     3     6




B =


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

```
C =

     8     9     7
     4     5     6
     3     1     2

D =

     3     1     2
     4     5     6
     8     9     7

E =

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

Reshaping/Resizing

```
X=[1 2 3 4 ; 2 3 4 5 ; 5 6 7 8 ];
display(X);

Y=reshape(X,2,6);
display(Y);

Z=reshape(X,6,2);
display(Z);
```

OUTPUT:

```
X =

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



Y =

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

```
Z =

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

Rotation

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

C=rot90(X);
display(C);
```

OUTPUT :

```
A =

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

C =

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

Flipping

```
A=[1 2 3 4];
display(A);

C=fliplr(A);
display(C);
```

OUTPUT:

```
A =

     1     2     3     4

C =

     4     3     2     1
```

```
A=[1 2 3 4; 2 3 4 5 ];
display(A);

C=fliplr(A);
display(C);
```

OUTPUT:

```
A =

     1     2     3     4
     2     3     4     5

C =

     4     3     2     1
     5     4     3     2
```

Logical Operations

```
A=[1 2 3 4 5];
display(A);
B=[1 0 0 1 0];
display(B);

C=xor(A,B);
display(C);
```

OUTPUT :

```
A =

     1     2     3     4     5

B =

     1     0     0     1     0

C =

     0     1     1     0     1
```

```
A=[1 0 1 0 1];
display(A);
B=[1 0 0 1 0];
display(B);

C=or(A,B);
display(C);

C=and(A,B);
display(C);

C=not(A);
display(C);
```

OUTPUT:

```
A =

     1     0     1     0     1

B =

     1     0     0     1     0

C =
        1     0     1     1     1

C =

        1     0     0     0     0

C =

        0     1     0     1     0
```

Relational Operators

```
A=[1 2 3; 4 5 6; 7 8 9];
B=[7 8 9; 4 5 6; 1 2 3];
display(A);
display(B);

display(A>B);
display(A<B);
display(A==B);
display(A~=B);
```

OUTPUT:

```
A =

    1    2    3
    4    5    6
    7    8    9



B =

    7    8    9
    4    5    6
    1    2    3

ans =

    0    0    0
    0    0    0
    1    1    1

ans =
     1     1     1
     0     0     0
     0     0     0

ans =

     0     0     0
     1     1     1
     0     0     0

ans =

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

EXPERIMENT 3

AIM-a) To generate Random Sequence and plot them
b) To calculate sum matrix, cumulative sum matrix and plot the sum matrix.

TOOLS USED- Matlab 7.0

THEORY:

When you create random numbers using software, the results are not random in a strict, mathematical sense. However, software applications, such as MATLAB®, use algorithms that make your results appear to be random and independent. The results also pass various statistical tests of randomness and independence. These apparently random and independent numbers are often described as *pseudorandom* and *pseudoindependent*. You can use these numbers as if they are truly random and independent. One benefit of using pseudorandom, pseudoindependent numbers is that you can repeat a random number calculation at any time. This can be useful in testing or diagnostic situations.
Although repeatability can be useful, it is possible to accidentally repeat your results when you really want different results. There are several ways to avoid this problem. The documentation contains several examples that show how to ensure that your results are different when that is your intention.

All the random number functions, rand, randn, randi, and randperm, draw values from a shared random number generator. Every time you start MATLAB®, the generator resets itself to the same state. Therefore, a command such as rand(2,2) returns the same result any time you execute it immediately following startup. Also, any script or function that calls the random number functions returns the same result whenever you restart.

PROCEDURE:

There are four fundamental random number functions: rand, randi, randn, and randperm. 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 indicates a fairly uniform sampling of numbers.
The randi function returns double integer values drawn from a discrete uniform distribution. For example,

```
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 of these values is roughly flat, which indicates a fairly uniform sampling of integers between 1 and 10.
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 r3looks like a roughly normal distribution whose mean is 0 and standard deviation is 1.
You can use the randperm function 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 behavior is useful for creating several different arrays of random values.

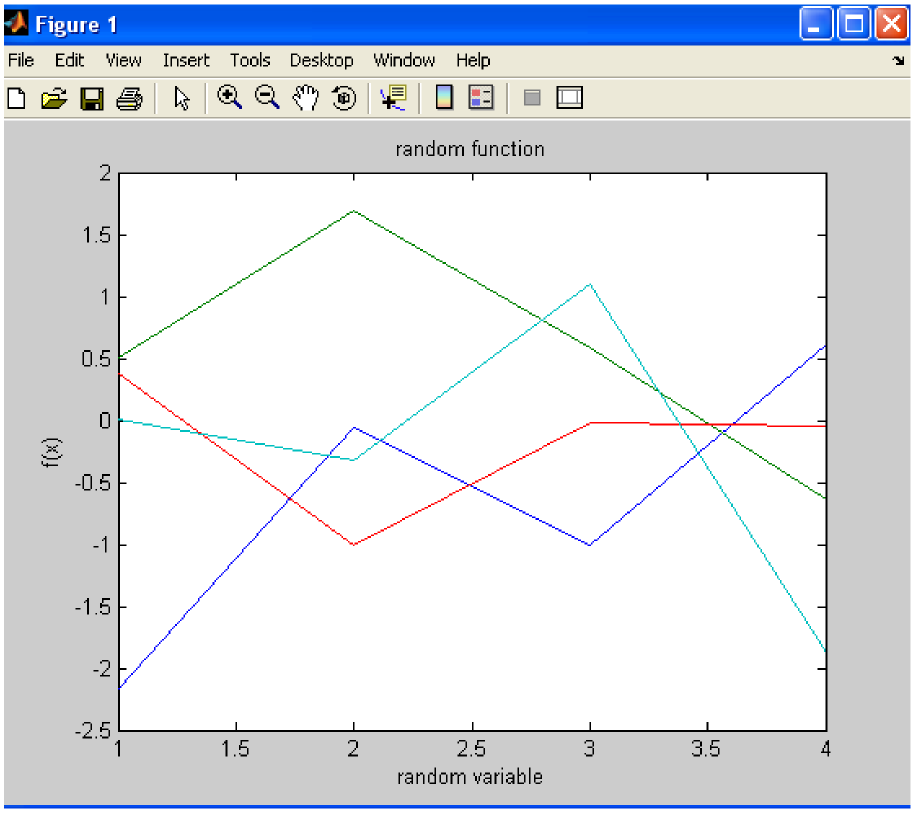
SUGGESTED SUGGESTED PROGRAM:

a)

1)

```
X=rand(4,4);
disp(x);
plot(x);
title('Random Sequence');
xlabel(' Random variable');ylabel('f(x)');
```

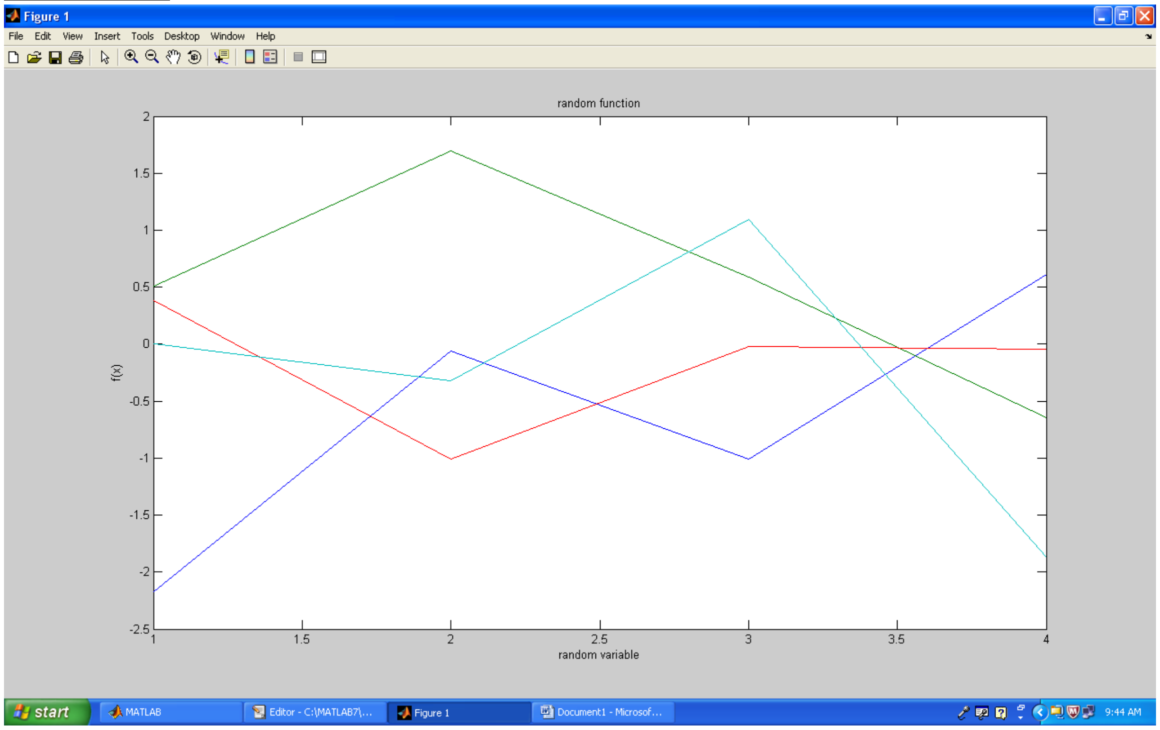
OUTPUT



2)

```
Z=randn(4,4);

disp(z);

plot(z)

title('random function');

xlabel('random variable');

ylabel('f(x)');
```

OUTPUT



b)

A=magic(4,4);

X=cumsum(A,1);

Y=cumsum(A,2);

Z=sum(A);

V=sum(A,1);

U=sum(A,2);

EXPERIMENT 4

AIM-a) 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.

TOOLS USED- Matlab 7.0

THEORY:

ceil(A) rounds the elements of A to the nearest integers greater than or equal to A. For complex A, the imaginary and real parts are rounded independently.
Y = round(X) rounds the elements of X to the nearest integers. For complex X, the imaginary and real parts are rounded independently.
Examples

```
a = [-1.9, -0.2, 3.4, 5.6, 7.0, 2.4+3.6i]

a =
  Columns 1 through 4
 -1.9000         -0.2000          3.4000          5.6000
  Columns 5 through 6
  7.0000          2.4000 + 3.6000i

round(a)

ans =
  Columns 1 through 4
 -2.0000               0          3.0000          6.0000
  Columns 5 through 6
  7.0000
```

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.
Examples

```
a = [-1.9, -0.2, 3.4, 5.6, 7.0, 2.4+3.6i]
```

```
a =
  Columns 1 through 4
  -1.9000           -0.2000             3.4000             5.6000

  Columns 5 through 6
   7.0000             2.4000 + 3.6000i
```

floor(a)

```
ans =
  Columns 1 through 4
  -2.0000           -1.0000             3.0000             5.0000

  Columns 5 through 6
   7.0000             2.0000 + 3.0000i
```

fix(A) rounds the elements of A toward zero, resulting in an array of integers. For complex A, the imaginary and real parts are rounded independently.
Examples

a = [-1.9, -0.2, 3.4, 5.6, 7.0, 2.4+3.6i]

```
a =
  Columns 1 through 4
  -1.9000       -0.2000       3.4000       5.6000

  Columns 5 through 6
   7.0000       2.4000 + 3.6000i
```

fix(a)

```
ans =
  Columns 1 through 4
  -1.0000       0             3.0000       5.0000

  Columns 5 through 6
   7.0000       2.0000 + 3.0000i
```

PROCEDURE:

To plot the graph of a function, you need to take the following steps:

1. Define **x**, by specifying the **range of values** for the variable **x**, for which the function is to be plotted.

2. Define the function, **y = f(x).**

3. Call the **plot** command, as **plot(x, y).**

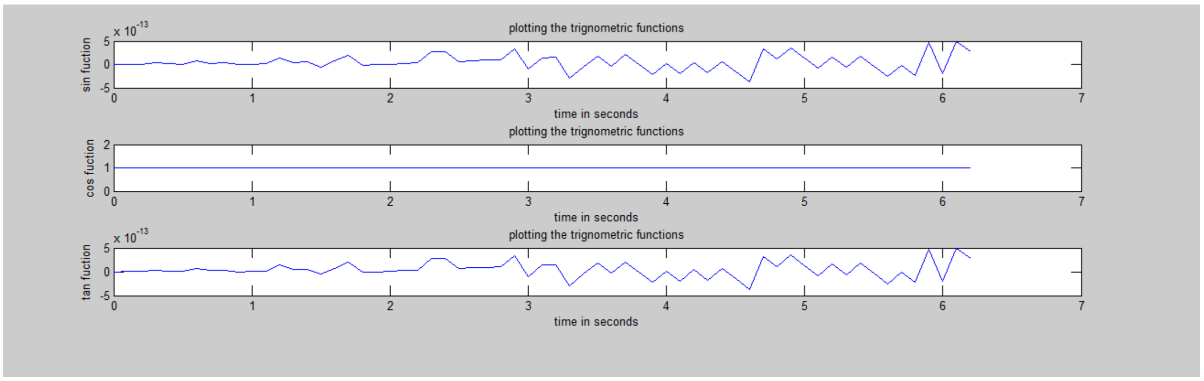
SUGGESTED PROGRAM:

a)

```
t=0:0.1:10*pi;
A=1; F=100;
x=A.*sin(2*pi*F*t);
subplot(511);
plot(t,x);
xlabel('time in seconds');
ylabel('sin fuction');
title('plotting the trignometric functions');

y=A.*cos(2*pi*F*t);
subplot(512);
plot(t,y);
xlabel('time in seconds');
ylabel('cos fuction');
title('plotting the trignometric functions');

r=A.*tan(2*pi*F*t);
subplot(513);
plot(t,r);
xlabel('time in seconds');
ylabel('tan fuction');
title('plotting the trignometric functions');

z=A.*cosec(2*pi*F*t);
subplot(514);
plot(t,z);
xlabel('time in seconds');
ylabel('cosec fuction');
title('plotting the trignometric functions');
```

```
q=A.*cot(2*pi*F*t);
subplot(515);
plot(t,q);
xlabel('time in seconds');
ylabel('cot fuction');
title('plotting the trignometric functions');

e=A.*sec(2*pi*F*t);
subplot(516);
plot(t,e);
xlabel('time in seconds');
ylabel('sec fuction');
title('plotting the trignometric functions');
```

OUTPUT

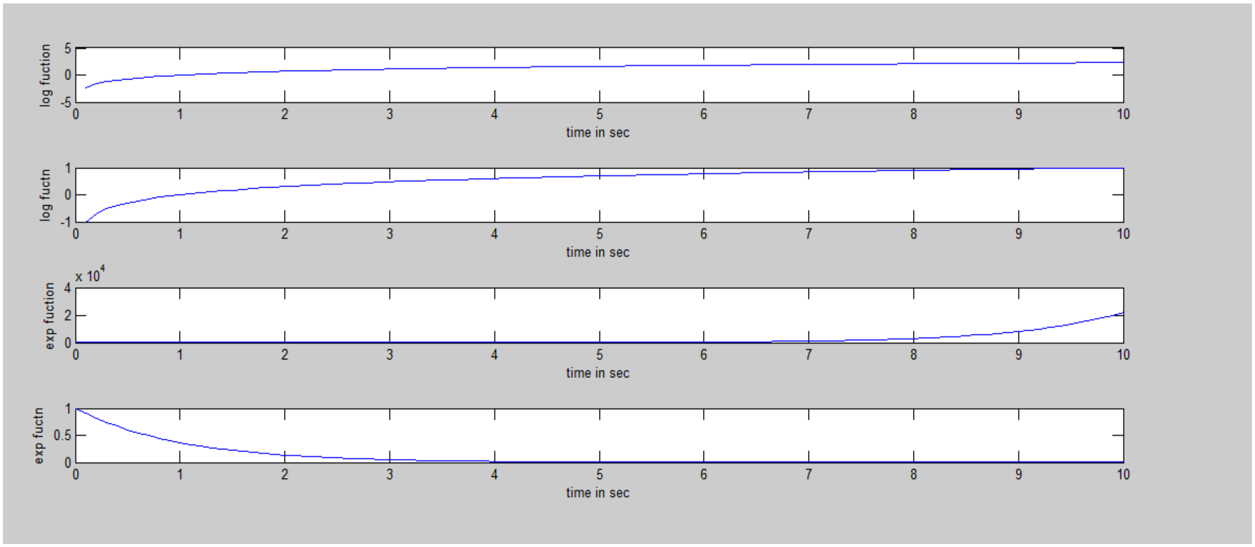


b)

```
x=0:0.1:10;
y=log(x);
title('log and exponential functions');
subplot(511);
plot(x,y);
xlabel('time in sec');
ylabel('log fuction');

z=log10(x);
subplot(512);
plot(x,z);
xlabel('time in sec');
```

```
ylabel('log fuctn');

u=exp(x);
subplot(513);
plot(x,u);
xlabel('time in sec');
ylabel('expfuction');

v=exp(-x);
subplot(514);
plot(x,v);
xlabel('time in sec');
ylabel('expfuctn');
```

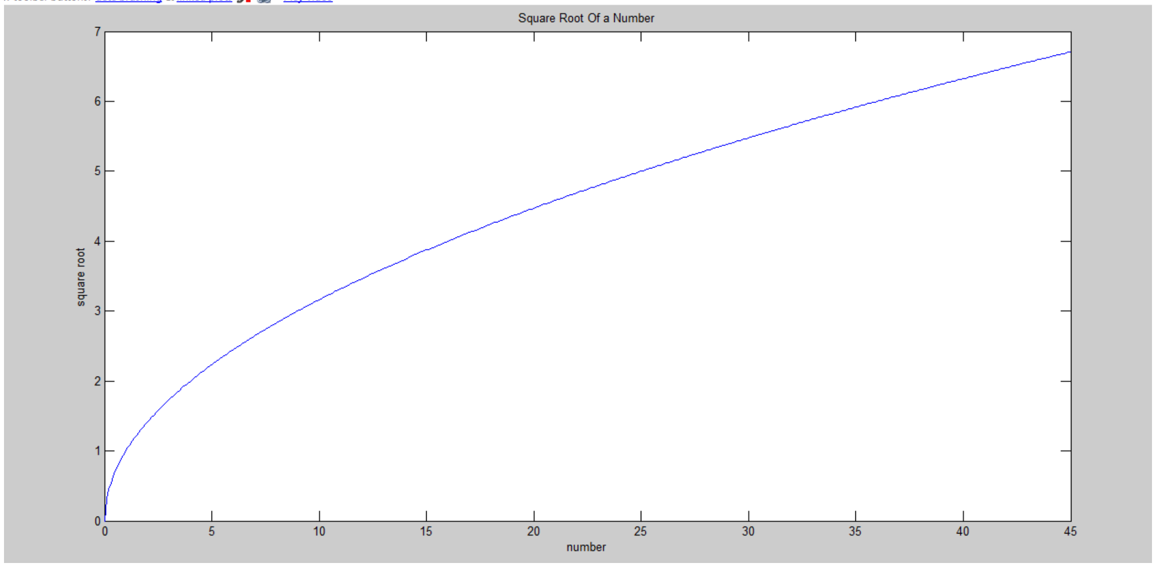
OUTPUT



c)

```
A=0:0.1:45;
z=sqrt(A);
disp(A);
disp(z);
```

OUTPUT


Square Root Of a Number
square root
number
0
1
2
3
4
5
6
7
0
5
10
15
20
25
30
35
40
45

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

TOOLS USED- Matlab 7.0

THEORY:

The exp function is an elementary function that operates element-wise on arrays. Its domain includes complex numbers.Y = exp(X) returns the exponential for each element of X. For complex , it returns the complex exponential

PROCEDURE

To plot the graph of a function, you need to take the following steps:

1. Define **x**, by specifying the **range of values** for the variable **x**, for which
    i. the function is to be plotted.

2. Define the function, **y = f(x).**

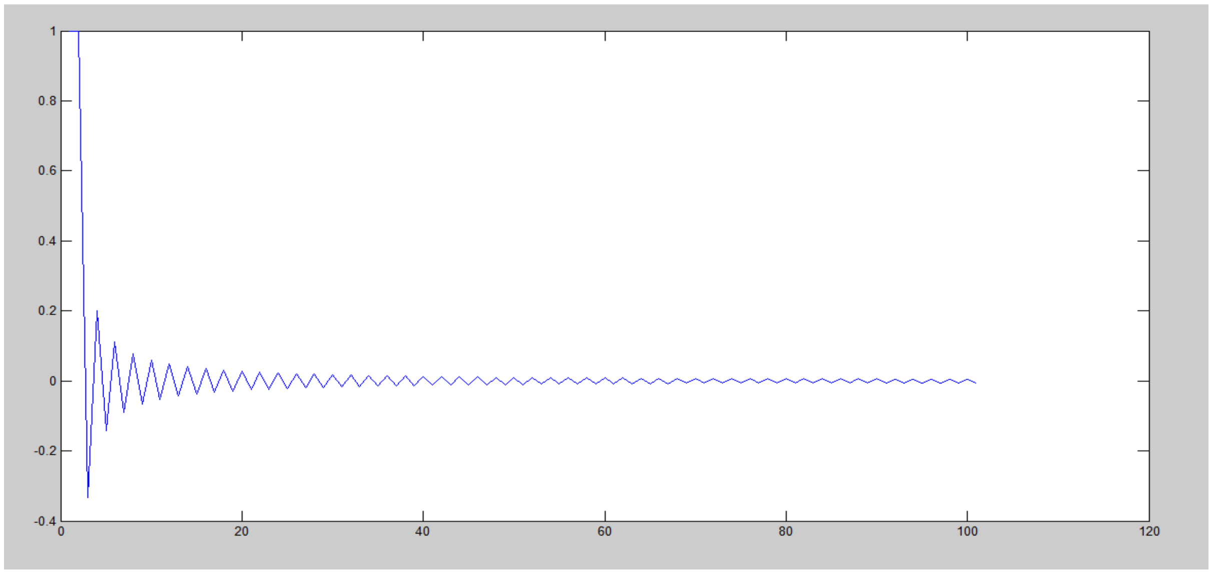
3. Call the **plot** command, as **plot(x, y).**

SUGGESTED PROGRAM:

a)

```
n=0:100;
Xn=[(-1).^(n+1)]./(2*n-1);
y=Xn;
sum(y);
disp(y);
plot(y);
```

OUTPUT

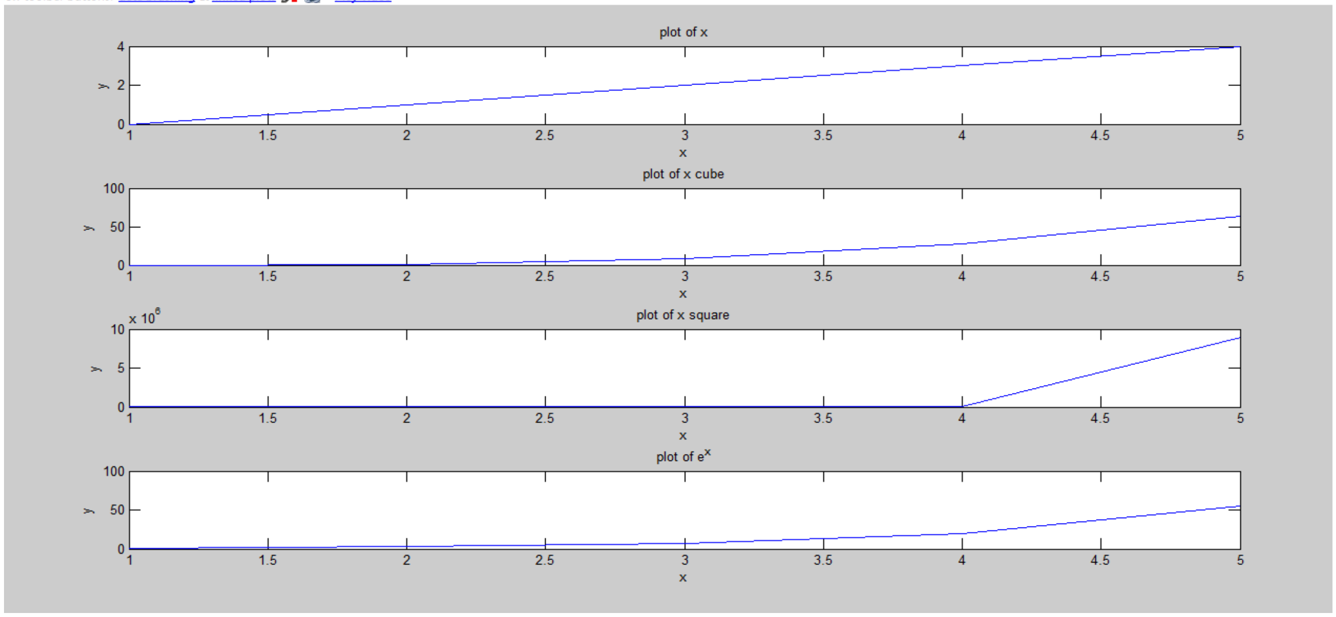


b)

```
x=0:1:4;

subplot(411);
plot(x);
title('plot of x');
xlabel('x');
ylabel('y');

y=x.^3;
subplot(412);
plot(y);
title('plot of x cube');
xlabel('x');
ylabel('y');


z=exp(x.^2);
subplot(413);
plot(z);
title('plot of x square');
xlabel('x');
ylabel('y');
```

```
v=exp(x);
subplot(414);
plot(v);
title('plot of e^x');
xlabel('x');
ylabel('y');
```

OUTPUT



EXPERIMENT 6

AIM-a) Generating a Sinusoidal Signal of a given frequency (say, 100Hz) and Plotting with Graphical Enhancements - Titling, Labeling, Adding Text, Adding Legends, Adding New Plots to Existing Plot, Printing Text in Greek Letters, Plotting as Multiple and Subplot.

TOOLS USED- Matlab 7.0

THEORY:

The sin function operates element-wise on arrays. The function's domains and ranges include complex values. All angles are in radians.
Y = sin(X) returns the circular sine of the elements of X.

MATLAB allows you to add title, labels along the x-axis and y-axis, grid lines and also to adjust the axes to spruce up the graph.

1. The **xlabel** and **ylabel** commands generate labels along x-axis and y-axis.
2. The **title** command allows you to put a title on the graph.
3. The **grid on** command allows you to put the grid lines on the graph.
4. The **axis equal** command allows generating the plot with the same scale factors and the spaces on both
5. axes.
6. The **axis square** command generates a square plot.

   Setting Colors on Graph

   MATLAB provides eight basic color options for drawing graphs. The following table shows the colors and their codes:

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

Setting Axis Scales

The axis command allows you to set the axis scales. You can provide minimum and maximum values for x and y axes using the axis command in the following way:

```
axis ( [xmin xmax ymin ymax] )
```

Generating Sub-Plots

When you create an array of plots in the same figure, each of these plots is called a subplot. The**subplot** command is for creating subplots.

Syntax for the command is:

```
subplot(m, n, p)
```

where, *m* and *n* are the number of rows and columns of the plot array and *p* specifies where to put a particular plot.

PROCEDURE

To plot the graph of a function, you need to take the following steps:

1. Define **x**, by specifying the **range of values** for the variable **x**, for which
   i. the function is to be plotted.

2. Define the function, **y = f(x).**

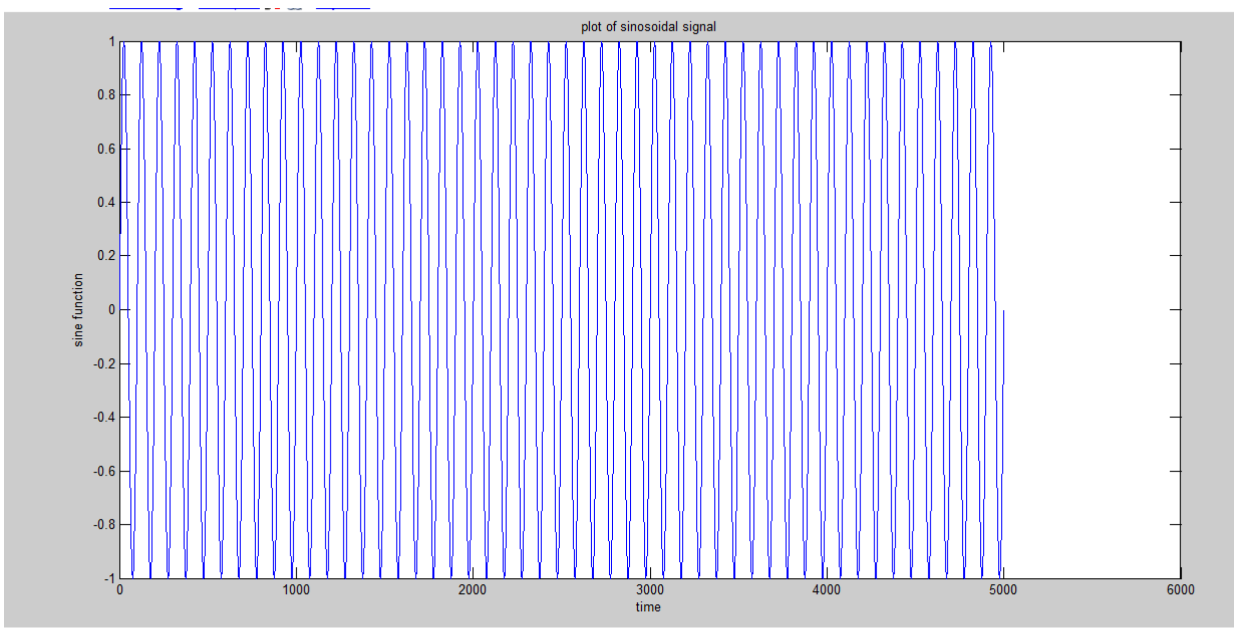
3. Call the **plot** command, as **plot(x, y).**

SUGGESTED PROGRAM:

a)

```
t=0:0.0001:.5;
a=1;
f=100;
x=a.*sin(2*pi*f*t);
plot(x);
xlabel('time');
ylabel('sine function');
title('plot of sinosoidal signal');
```

<u>OUTPUT</u>


plot of sinosoidal signal
sine function
time
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1000
2000
3000
4000
5000
6000

EXPERIMENT 7

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

TOOLS USED- Matlab 7.0

THEORY:

MATLAB provides the **dsolve** command for solving differential equations symbolically.
The most basic form of the **dsolve** command for finding the solution to a single equation is :

```
dsolve('eqn')
```

where *eqn* is a text string used to enter the equation.

It returns a symbolic solution with a set of arbitrary constants that MATLAB labels C1, C2, and so on.

You can also specify initial and boundary conditions for the problem, as comma-delimited list following the equation as:

```
dsolve('eqn','cond1', 'cond2',…)
```

For the purpose of using dsolve command, **derivatives are indicated with a D**. For example, an equation like f'(t) = -2*f + cost(t) is entered as:
'Df = -2*f + cos(t)'

Higher derivatives are indicated by following D by the order of the derivative.

For example the equation f''(x) + 2f'(x) = 5sin3x should be entered as:

'D2y + 2Dy = 5*sin(3*x)'

Let us take up a simple example of a first order differential equation: y' = 5y.

```
s = dsolve('Dy = 5*y')
```

MATLAB executes the code and returns the following result:

```
s =
 C2*exp(5*t)
```

PROCEDURE:

To plot the graph of a function, you need to take the following steps:

1. Define **x**, by specifying the **range of values** for the variable **x**, for which
    i. the function is to be plotted.

2. Define the function, **y = f(x).**

3. Call the **plot** command, as **plot(x, y).**

SUGGESTED PROGRAM:

```
dsolve('Dy=y*x);
```

OUTPUT

```
C1*exp(x*t)
```

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

$$h(T) = (T - 10) \text{ for } 0 < T < 100$$

$$= (0.45\,T + 900) \text{ for } T > 100.$$

Exercise : Testing the Scripts written using A). T = 5, h = -5 and B). T = 110, h =949.5

TOOLS USED- Matlab 7.0

THEORY:

Program files can be *scripts* that simply execute a series of MATLAB® statements, or they can be *functions* that also accept input arguments and produce output. Both scripts and functions contain MATLAB code, and both are stored in text files with a .m extension. However, functions are more flexible and more easily extensible.
MATLAB expression, usually consist of variables or smaller expressions joined by relational operators (e.g., count < limit), or logical functions (e.g., isreal(A)). Simple expressions can be combined by logical operators (&&, ||, ~) into compound expressions such as the following. MATLAB evaluates compound expressions from left to right, adhering to operator precedence rules.

(count < limit) && ((height - offset) >= 0)

Nested if statements must each be paired with a matching end. The if function can be used alone or with the else and elseif functions. When using elseif and/or else within an if statement, the general form of the statement is

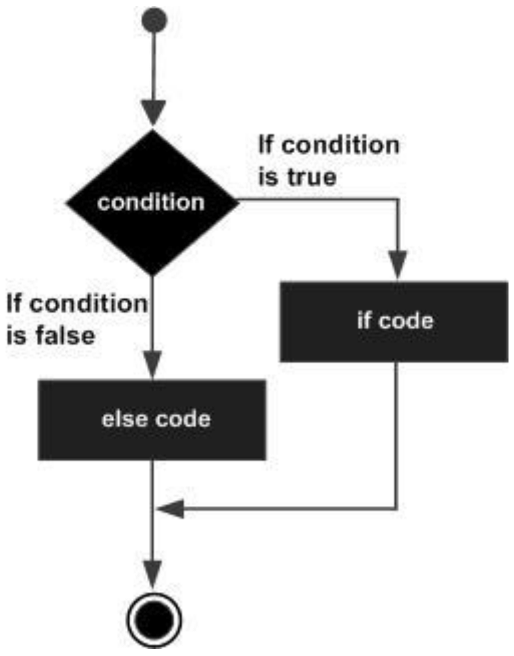
```
if expression1

   statements1

elseif expression2

   statements2

else

   statements3

end
```

PROCEDURE:

An **if** statement can be followed by an optional **else if...else** statement, which is very useful to test various conditions using single if...else if statement.

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

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



SUGGESTED PROGRAM:

```
clc
T=input('enter the value of T for the function h(T)');
If(0<T&&T<100)
display('value of h');
h=T-10;
elseif(T>100) display('value of h')
h=0.45*T;
h=h+900;
elseif(T<0)
T=0;
else
display('error')
```

```
end
```

<u>OUTPUT</u>

```
enter the value of T for the function h(T) 15
T=15
Value h
h=5
```

EXPERIMENT 9

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

TOOLS USED- Matlab 7.0

THEORY:

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.

The three pictures on the right demonstrate the phenomenon for a square wave (of height $\pi/4$) whose Fourier expansion is

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

More precisely, this is the function *f* which equals $\pi/4$ between $2n\pi$ and $(2n+1)\pi$ and $-\pi/4$ between $(2n+1)\pi$ and $(2n+2)\pi$ for every integer *n*; thus this square wave has a jump discontinuity of height $\pi/2$ at every integer multiple of π.

As can be seen, as the number of terms rises, the error of the approximation is reduced in width and energy, but converges to a fixed height. A calculation for the square wave (see Zygmund, chap. 8.5., or the computations at the end of this article) gives an explicit formula for the limit of the height of the error. It turns out that the Fourier series exceeds the height $\pi/4$ of the square wave by

$$\frac{1}{2}\int_0^\pi \frac{\sin t}{t}\,dt - \frac{\pi}{4} = \frac{\pi}{2}\cdot(0.089490\ldots)$$

or about 9 percent. More generally, at any jump point of a piecewise continuously differentiable function with a jump of *a*, the *n*th partial Fourier series will (for *n* very large) overshoot this jump by approximately $a\cdot(0.089490\ldots)$ at one end and undershoot it by the same amount at the other end; thus the "jump" in the partial Fourier series will be about 18% larger than the jump in the original function. At the location of the discontinuity itself, the partial Fourier series will converge to the midpoint of the jump (regardless of what the actual value of the original function is at this point). The quantity

$$\int_0^\pi \frac{\sin t}{t}\,dt = (1.851937052\ldots) = \frac{\pi}{2} + \pi\cdot(0.089490\ldots)$$

is sometimes known as the *Wilbraham–Gibbs constant.*

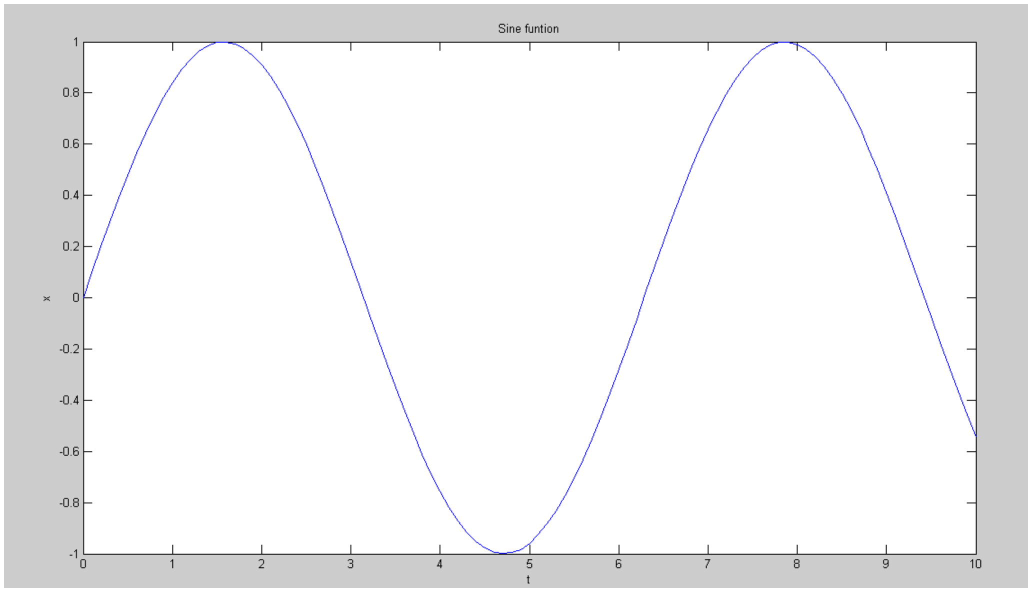
PROCEDURE:

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 add the third harmonic to the fundamental, and plot it.
4. Now use the first, third, fifth, seventh, and ninth harmonics.
5. 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.
6. These vectors are plotted on the same figure to show the evolution of the square wave. Note Gibbs' effect.

SUGGESTED PROGRAM:

```
t=0:0.1:10;

x=sin(t);

plot(t,x);

title('Sine funtion');

xlabel('t');

ylabel('x');
```

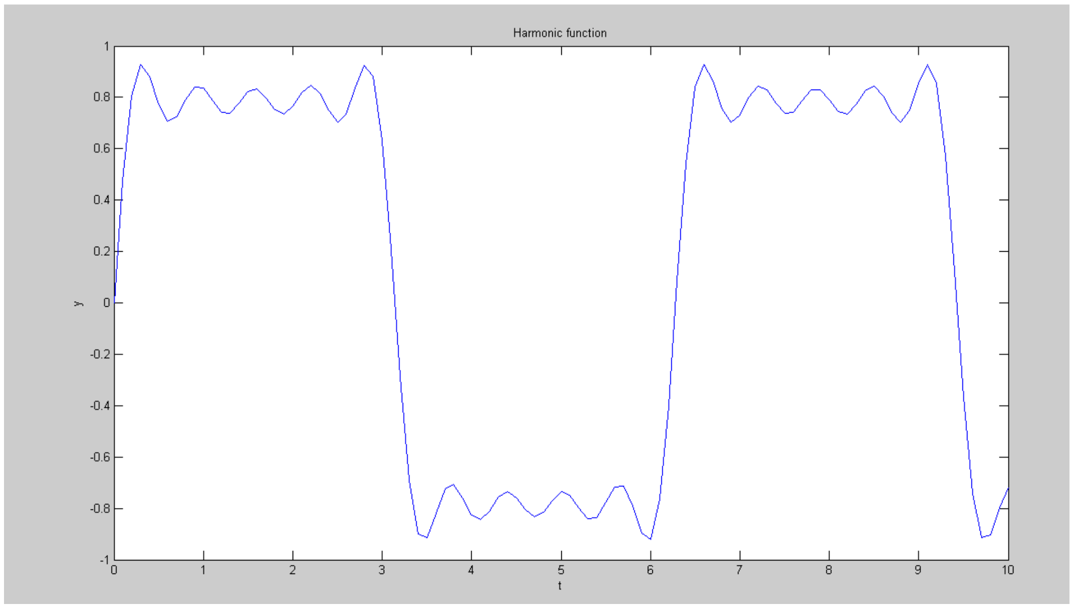
OUTPUT



b)

```
f=5;

w=2*pi*f;

t=0:0.0001:1;

y=0;

for n=1:2:99;

y=y+(1/n).*sin(n*w*t);

end

plot(t,y);

title('Harmonic funtion');

xlabel('x');

ylabel('y');
```

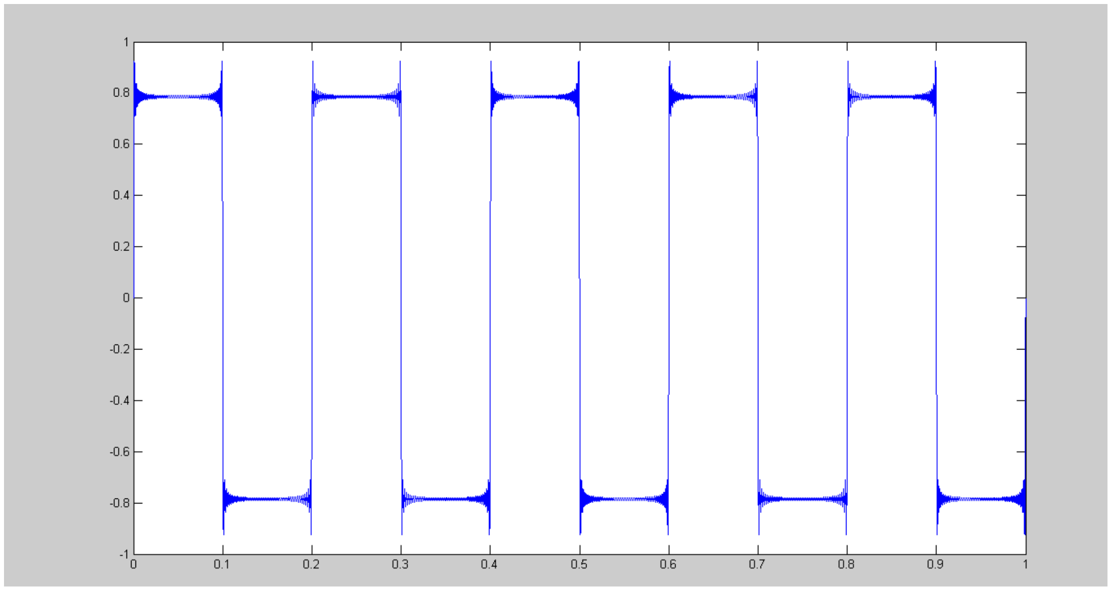
<u>OUTPUT</u>



c)

```
f=5;
w=2*pi*f;
t=0:0.0001:1;
y=0;
for n=1:2:99;
    y=y+(1/n).*sin(n*w*t);
end
plot(t,y);
```

<u>OUTPUT</u>


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1