LIST OF EXPERIMENTS

NUMERICAL TECHNIQUES LAB

1. Study of Introduction to MATLAB
2. Study of basic matrix operations
3. To solve linear equation
4. Solution of Linear equations for Underdetermined and Overdetermined cases.
5. Determination of Eigen values and Eigen vectors of a Square matrix.
6. Solution of Difference Equations.
7. Solution of Difference Equations using Euler Method.
8. Solution of differential equation using 4^{th} order Runge- Kutta method.
9. Determination of roots of a polynomial.
10. Determination of polynomial using method of Least Square Curve Fitting.
11. Determination of polynomial fit, analyzing residuals, exponential fit and error bounds from the given data.
12. Determination of time response of an R-L-C circuit.

EXPERIMENT NO.1

OBJECTIVE: STUDY OF INTRODUCTION TO **MATLAB**

1. INTRODUCTION:

The name MATLAB stands for Matrix Laboratory. The basic building block of MATLAB is the matrix. It is not confined to the solution of Matrix related problems. With its inbuilt functions, it is an excellent tool for linear algebraic computations, data analysis, signal processing, optimization, numerical solutions of ordinary differential equations (ODE), quadrature, 2D & 3D, graphics and many other types of scientific computation. Therefore, we can say:

MATLAB is a software package in 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:

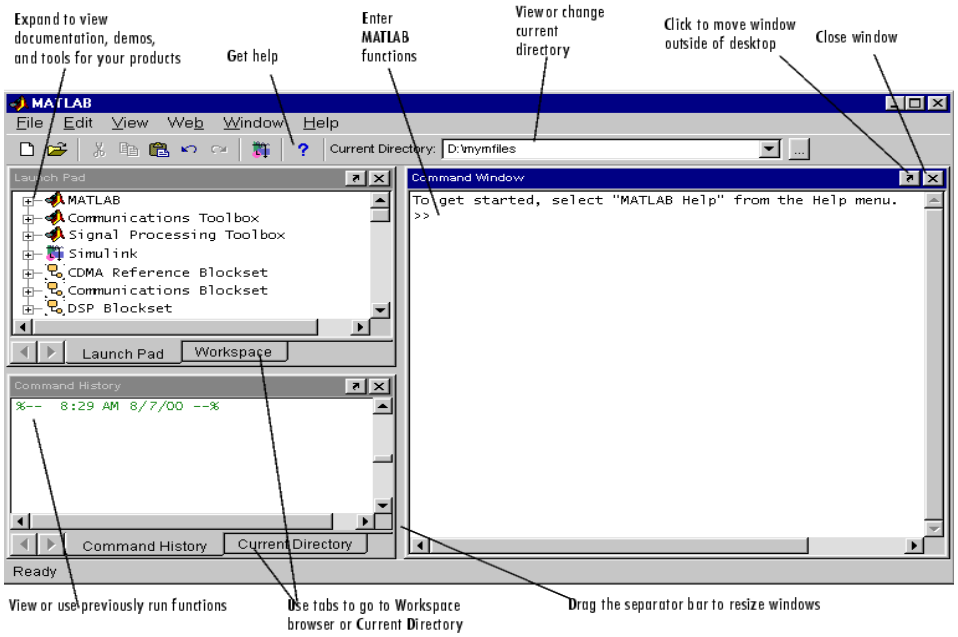
- Math and computation
- Algorithm and development
- Data acquisition modeling
- Simulation and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building.

2. BASICS:

2.1 MATLAB WINDOWS: There are three basic window which are as follows:

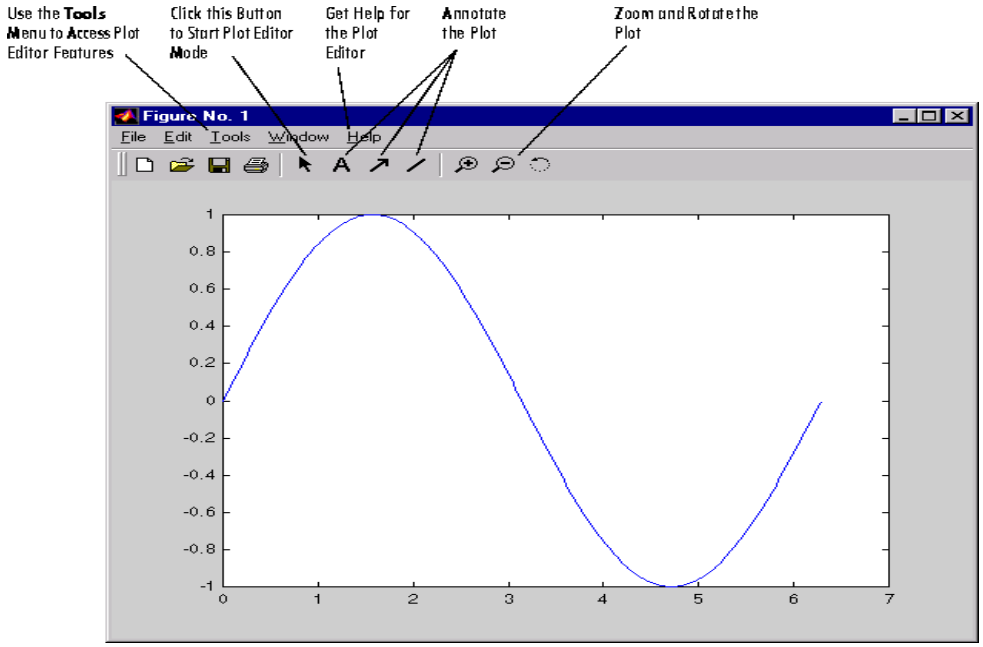
1. **MATLAB DESKTOP:** This is the window which appears by default when MATLAB is launched. It consists of :

- COMMAND WINDOW**:** This is the main window , where command are written. It is characterized by MATLAB command prompt (>>). The results also appear on this window(except figures, which appear on figure window) which command is written. Commands cannot be edited in this window.
- CURRENT DIRECTORY: This appears on the bottom left side of MATLAB desktop. It is where all files are listed. With a mouse right click, you can run M-files, rename, delete them etc after selecting a file from here.
- WORKSPACE: This sub-window shows all the variables generated so far and also shows their type and size.
- COMMAND HISTORY: All commands typed on the MATLAB prompt are recorded here. Also commands can be selected from here and create as M-file. Thus it remains records of MATLAB functions run.



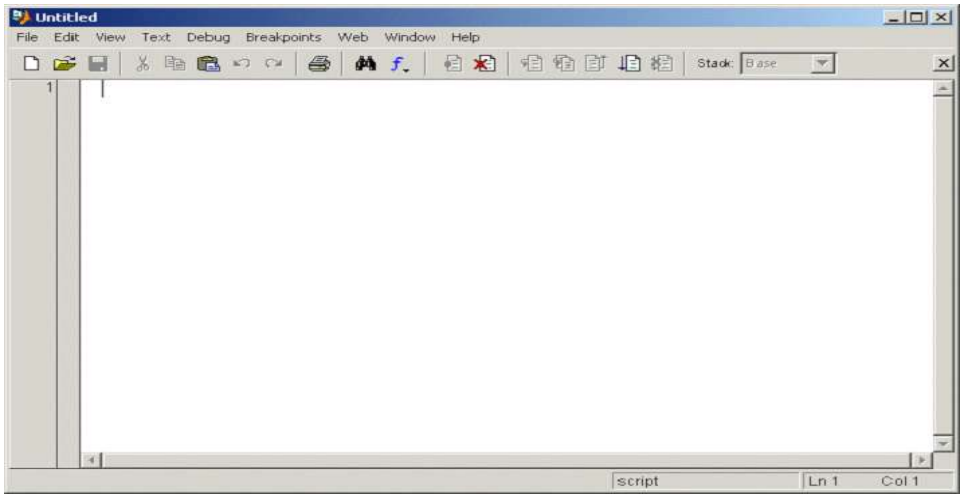
2. FIGURE WINDOW:

The output of all the commands written on the command window or executed by writing in M-file, whose output is a graph , appears on the window. The user can create as many figure windows as the system memory allows. A figure window is showing sine curve is shown in figure.



3. EDITOR WINDOW:

This is where we can write, create, edit and save programs in a file. The file is known as M-file. To select editor window, go to file and then select M-file. The programmes written on the file are first saved and then run to get the results. To save and run the programme, go to debug and select ‘save and run’. The result appear on the command window. The figure appear on the figure window. A editor window is shown as in figure:



2.2 LANGUAGES:

MATLAB is a high-level language that includes matrix-based data structures, its own internal data types, an extensive catalog of functions and scripts, the ability to import and export to many types of data files, object-oriented programming capabilities, and interfaces to external technologies such as COM, Java, program written in C and Fortran, and serial port devices.

2.3 INPUT-OUTPUT:

MATLAB takes input from the screen and rushes output to the screen i.e. it supports interactive computation. It can read input files and write output files.

2.3.1 Data Type: There is no need to declare a data to be real or complex. When a real number is entered as a variable, MATLAB automatically sets the variable to be real. Fundamental data type is the array.

2.3.2 Dimension: Dimension state is not required in the MATLAB. It is automatic.

2.3.3 Case Sensitivity: It differentiates between lower and upper cases. It is case sensitive.

2.3.4 Output Display: The output of every command appears on the screen unless MATLAB is directed otherwise. A semi-colon(;) suppress the output when used at the end of a command except for the graphics.

2.4 GETTING STARTED

2.4.1 STARTING MATLAB: On windows platform, start MATLAB by double clicking the MATLAB shortcut icon on your Windows desktop.

2.4.2 WRITING COMMAND: When you start MATLAB, the MATLAB desktop appears containing tools for managing files, variables, and applications associated with MATLAB. You can start writing your command at the prompt appears on command Window. You can also write command in M-file.

2.4.3 PRINTING GRAPHICS: The simplest way to get print out of the graph is to type print command window after the graph appears in the figure window. Alternatively activate the figure window and then select print from the file menu.

2.4.4 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 that, for example, executes function to save the workplace, or display a quit confirmation dialog box.

3. ACCESSORIES

3.1 TOOLBOXES: MATLAB features a family of add-on-application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others. There are 78 toolboxes available. Thus we can say MATLAB basically works as a platform and for solving a particular problem, concerned toolbox is required. Few of the toolboxes are as follows:

Communication toolbox

Control system toolbox

Curve fitting toolbox

Data acquisition toolbox

Filter design toolbox

Fuzzy logic toolbox

Instrument control toolbox

Optimization toolbox

Statistics toolbox

Symbolic maths toolbox, etc.

3.2 SIMULINK: Simulink is a software package that enables you to model, simulate, and analyze systems whose outputs change over time. Such system are often referred to as dynamic systems. Simulink can be used to explore the behavior of a wide range of real- world dynamic systems, including electrical circuits, shock

absorbers, braking systems, and many other electrical, mechanical, and thermodynamic system. This section explains how Simulink works.

Simulating a dynamic system is a two step process with Simulink. First, a user creates a block diagram, using a block diagram Simulink model editor that graphically depicts time dependent mathematical relationship among the system's inputs, states, and outputs. The user then commands simulink to simulate the system represented by the model from a specified start time to a specified stop time.

EXPERIMENT NO. 2

OBJECTIVE: STUDY OF BASIC MATRIX OPERATIONS

APPARATUS/ SOFTWARE REQUIRED:

SR.NO	APPARATUS/SOFTWARE	SPECIFICATION	QUANTITY

GIVEN NUMERICAL:

A= 3 2 1
0 3 4
-1 1 -1

B= 1 3 0
2 6 4
-1 0 2

Where A and B are two 3 X 3 matrix

COMMANDS:

>> A= [3 2 1; 0 3 4; -1 1 -1]
>> B= [1 3 0; 2 6 4; -1 0 2]

Sum: A+ B
Subtraction: A – B
Multiplication: A * B
Division: A/B or A\B
Inverse of A: inv(A)

RESULTS:

>> A + B **% Sum up matrix A and B**

ans:

```
    4     5     1
    2     9     8
   -2     1     1
```

>> A – B **% Subtract matrix A and B**

ans:

```
    2    -1     1
   -2    -3     0
    0     1    -3
```

>> A * B **% Multiply matrix A and B**

ans:

```
    6    21    10
    2    18    20
    2     3     2
```

>> A/B **% Divides matrix A and B(take inverse of B and multiply with A)**

ans:

```
-2.1667    1.4167   -2.3333
      0    0.5000    1.0000
   2.1667   -0.9167    1.3333
```

>> A\B **% Divides matrix A and B(take inverse of A and multiply with B)**

ans:

```
    0.2308    0.1154   -0.8462
   -0.1538    0.9231    1.2308
    0.6154    0.8077    0.0769
```

>> inv(A) **% inverse of A**

ans:

0.2692	**-0.1154**	**-0.1923**
0.1538	**0.0769**	**0.4615**
-0.1154	**0.1923**	**-0.3462**

ADDITIONAL COMMANDS AND RESULTS:

>>a= magic(3) **% gives 3 X 3 matrix whose sum from any angle is same**

ans:

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

>>a= rand(3) **% gives any 3 X 3 random matrix**

ans:

0.8147	**0.9134**	**0.2785**
0.9058	**0.6324**	**0.5469**
0.1270	**0.0975**	**0.9575**

>> a= ones(3) **% gives 3X 3 matrix whose elements are one**

ans:

1	**1**	**1**
1	**1**	**1**
1	**1**	**1**

>> b= 2*ones(3) **% multiplication of 2 with ones(3)**

ans:

2 2 2
2 2 2
2 2 2

>> a+2 **% summation of 2 with matrix A**

ans:

>> a(2,2) **% gives second row and second column element of matrix A**

ans:

1

>>a(2:3, :) **%gives second and third row of matrix 'a'**

ans:

1 1 1
1 1 1

>> a(:, 2:3) **% gives second and third column of matrix 'a'**

ans:

1 1
1 1
1 1

>> a(2:3,1:2) **%gives second and third row and first and second column of matrix 'a'**

ans:

1 1

1 1

>>a(:,2) **% gives second column of matrix 'a'**

ans:

1

1

1

>>a(:,1:2)=[] **%delete first and second column**

ans:

1

1

1

>>eye(3) **%gives 3 X 3 matrix whose diagonal are one**

ans:

1 0 0

0 1 0

0 0 1

>> diag(a) **% gives diagonal element of matrix 'a'**

ans:

1 0 0

0 1 0

0 0 1

>> b = a' **% gives inverse of matrix 'a'**

ans:

1 1 1

CONCLUSION:

We have studied various commands to solve the basic matrix operations.

PRE- EXPERIMENTAL QUESTION

Q1. What is a Matrix?

Q2. What is the dimension of a matrix?

Q3. What are the elements of a matrix?

Q4. What is a square matrix?

Q6. What are the basic matrix operations?

Q7. What is the Transpose of a matrix?

POST- EXPERIMENTAL QUESTION

Q8. What are the commands in MATLAB to solve the various basic matrix operations?

EXPERIMENT NO. 3

OBJECTIVE: TO SOLVE LINEAR EQUATION

APPARATUS/ SOFTWARE REQUIRED:

SR.NO	APPARATUS/SOFTWARE	SPECIFICATION	QUANTITY

THEORY:

LINEAR EQUATION:

Solving a linear algebraic equation is easy in MATLAB. It is, perhaps, also the most used computation in science and engineering. We will solve a set of linear algebraic equations given below:

$$5x= 3y-2z+10$$
$$8y+4z= 3x+20$$
$$2x+4y-9z=9$$

PROCEDURE:

STEP 1: Rearrange equations: Write each equation with all unknown quantities on the left hand side and all known quantities on the right hand side. Thus, for the equations given above, rearrange them such that all terms involving x, y, and z are on the left hand side of the equal sign:

$$5x-3y+2z = 10$$
$$-3x+8y+4z = 20$$
$$2x+4y-9z = 9$$

STEP 2: Write the equations in matrix form: To write the equation in the matrix form [A]{x}={b} where {x} is the vector of unknowns, we have to arrange the unknowns in matrix A and the constants on the right hand side of the equations in vector b.

In this particular example, the unknown vector is

$$x = \begin{bmatrix} x \\ y \\ z \end{bmatrix},$$

The coefficient matrix is

$$A = \begin{bmatrix} 5 & -3 & 2 \\ -3 & 8 & 4 \\ 2 & 4 & -9 \end{bmatrix},$$

And the known constant vector is

$$b = \begin{bmatrix} 10 \\ 20 \\ 9 \end{bmatrix}.$$

Note that the columns of A are simply the coefficients of each unknown from all the three equations.

STEP 3: Solve the matrix equation in MATLAB: Enter the matrix A and vector b, and solve for vector x with x=A\b (note that the \ is different from the division /):

```
>> A= [5   -3    2;   -3    8    4;   2    4    -9];          % enter matrix A
>> b=[10
      20
       9].;                                                  % enter column vector b
>> x= A\b                                                    % solve for x
x=
   3.4442                                                    the \ is used to solve a linear system
   3.1982                                                    of equations [A]{x}={b}.
   1.1868
>> c= A* x                                                   % check for solution
c=
    10.0000
    20.0000
     9.0000
```

Program: Write a program to solve linear equation

1 5x= 5y-8z

9y+3z= x+6

10x+4y-z=7

```
>>A=[15   -5  8; -1   9  3;10  4  -1];
>> b = [0
        6
        7]
>>x=A\b;
>> c=A*x
```

RESULT:

```
      x= 0.3688
         0.7764
        -0.2063
      c=  0.0000
         6.0000
         7.0000
```

CONCLUSION: Hence we solve the given linear equation with the help of MATLAB.

EXPERIMENT NO. 4

OBJECTIVE: SOLUTION OF LINEAR EQUATIONS FOR UNDER DETERMINED AND OVERDETERMINED CASES

APPARATUS/ SOFTWARE REQUIRED:

SR.NO	APPARATUS/SOFTWARE	SPECIFICATION	QUANTITY

1. GIVEN NUMERICAL

i) **Overdetermined :** Under such conditions, number of equations are more than the unknowns. Overdetermined systems of simultaneous linear equations are often encountered in various kinds of curve fitting to experimental data.

e.g.

x + 2y – z = 3

3x – y + 2z = 1

2x -2y +3z = 2

x- y + z = -1

ii) **Underdetermined:** Underdetermined linear systems involve more unknowns than equations. When they are accompanied by additional constraints, they are the purview of linear programming. By itself, the backslash operator deals only with the unconstraint system. The solution is never unique. MATLAB finds a basic solution, which has at most m nonzero components, but even this may not be unique. The particular solution actually computed is determined by the QR factorization with column pivoting. The complete solution of the underdetermined system can be characterized by adding an arbitrary vector from the null space, which can be found using the null function with an option requesting a

"rational" basis Z = null (a, 'r'). When the equations are expressed by a* x = b.

It can be confirmed that a*Z is zero and that any vector x, where

x = b +Z*q

for any arbitrary vector q satisfies a*x = b.

e.g.

x + 2y – z = 3

3x – y + 2z = 1

2. COMMANDS:

The numerical given are expressed in matrix form i.e. ax = b and the commands are as follows:

i) Overdetermined

```
>> a = [1   2  -1; 3  -1  2; 2  -2  3; 1  -1  1];
>> b = [3  ; 1; 2; -1];
>> x = a\b
```

ii) Underdetermined

```
>> a = [1  2  -1; 3  -1  2];
>> b = [3;1];
>> x = a\b
>> Z = null(a, 'r')
```

3. RESULTS:

i) Overdetermined

```
>> a = [1   2  -1; 3  -1  2; 2  -2  3; 1  -1  1];
                                  % write equations in matrix form
>> b = [3  ; 1; 2; -1];
>> x = a\b
x=    -1.0000
```

4.0000

4.0000

>> a* x %this verifies.

ans = 3.0000

1.0000

2.0000

-1.0000

ii) Underdetermined

>> a = [1 2 -1; 3 -1 2]; % write equations in matrix form

>> b = [3;1];

>> x = a\b

x = 0.7143

1.1429

0

>> a* x

ans=

3

1

4. **CONCLUSION:** Hence, we have found out the solution of underdetermined and over determined cases and then verified again.

PRE-EXPERIMENTAL QUESTIONS

Q1.What do you mean by underdetermined system?

Q2. What do you mean by over determined system?

Q3.How do you solve linear equations of under determined system and over determined system manually?

POST-EXPERIMENTAL QUESTIONS

Q4. What are the commands to solve linear equations of under determined system and over determined system in MATLAB?

EXPERIMET NO. 5

OBJECTIVE: DETERMINATION OF EIGEN VALUES AND EIGEN VECTORS OF A SQUARE MATRIX.

APPARATUS/ SOFTWARE REQUIRED:

SR.NO	APPARATUS/SOFTWARE	SPECIFICATION	QUANTITY

THEORY:

For an n x n matrix **A**, the real number λ is called an ***Eigen value*** of A if there exists a nonzero vector **x** in Rn such that A**x** =λ **x**. The vector **x** is called an eigenvector belonging to λ . The equation A**x** =λ **x** is equivalent to (A-λI) **x** = **0**, so all of the following are equivalent:

1.λ is an Eigen value of A.

2. (A-λI) **x** = **0** has a nontrivial solution.

3. A-λI is singular.

4. det(A-λI) = 0.

The eigenvectors for λ are the nonzero solutions **x** to (A-λI)**x**=**0**. These vectors together with the **0** vector is called the ***eigen space*** corresponding to eigen value λ. The expression det(A-λI) is a polynomial in λ of degree n, called the ***characteristic polynomial***. By property 4, the eigen values are the roots of the ***characteristic equation*** det(A-λI) = 0.

Determining eigen values and eigenvectors with MATLAB:

Method 1: In MATLAB we can find the characteristic polynomial of a matrix A by entering **poly(A)**. If A is an *n* x *n* matrix, **poly(A)** is a row vector with *n*+1 elements that are the coefficients of the characteristic polynomial. The command **roots(C)** computes the roots of the polynomial whose coefficients are the elements of the vector C. Thus, **roots (poly(A))** returns the eigen values of A in a column vector.

To find the eigenvectors corresponding to each eigenvalue found above, we need to find the nonzero solutions **x** to (A-I)**x** = **0.** One way of doing this in MATLAB is to compute **rref(A-I)** and then use Gauss-Jordan elimination.

Finding the eigenvector and eigenvalues of a square matrix A.

```
» A = [3 2 -2; -3 -1 3; 1 2 0]        Enter matrix A.

A =
    3    2   -2
   -3   -1    3
    1    2    0

» roots(poly(A))                     Compute the roots of characteristic equation
                                     det(A-λI)=0;  returns eigenvalues 2, 1, and -1.
ans =
   2.0000
   1.0000
  -1.0000

» rref(A-2*eye(3))                   Use rref to find the solutions to (A-2I)x = 0
                                     -- the eigenvectors corresponding to λ = 2.
ans =
    1    0    0
    0    1   -1
    0    0    0

» rref(A-1*eye(3))                   Use rref to find the solutions to (A-1I)x = 0
                                     -- the eigenvectors corresponding to λ = 1.
ans =
    1    0   -1
    0    1    0
    0    0    0

» rref(A-(-1)*eye(3))                Use rref to find the solutions to (A-(-1)I)x = 0
                                     -- the eigenvectors corresponding to λ = -1.
ans =
    1    0   -1
    0    1    1
    0    0    0
```

The reduced form of echelon form for A-2I gives the general solution to(A-2I)x =0

as $x=\begin{bmatrix}0\\ r\\ r\end{bmatrix}=r\begin{bmatrix}0\\1\\1\end{bmatrix}$, the form for eigenvectors corresponding to λ = 2.

Similarly, the reduced echelon forms for A-1I and A-(-1)I allow us to determine eigenvectors of the form s $\begin{bmatrix}1\\0\\1\end{bmatrix}$ for λ = 1, and t $\begin{bmatrix}1\\-1\\1\end{bmatrix}$ corresponding to λ = -1.

Method 2: Determining the **eigenvalues** and **eigenvectors** in **MATLAB** is to use the ***eig*** function. For an nxn matrix A, eig(A) returns a n x1 column vector whose elements are the eigenvalues of A. The command in the form

$$[V\ D]=eig(A)$$

Computes both the eigenvalues and eigenvectors of A. V will be a matrix whose columns are eigenvectors of A and D will be a diagonal matrix whose entries along the diagonal are eigenvalues of A. The ith column of V, V(:, i), is the eigenvector corresponding to the eigenvalue D(i,i). A sample session is shown for the matrix A above.

Finding the eigenvector and eigenvalues of a square matrix A.

```
» eig(A)                                  Compute the eigenvalues for A.
                                          Note same answer as with roots(poly(A)).
ans =
  -1.0000
   1.0000
   2.0000

» [V D] = eig(A)                          This form gives us both the eigenvectors in V
                                          and the eigenvalues in D.
V =                                       The columns of V are the eigenvectors with
  -0.5774   0.7071   0.0000               norm 1.
   0.5774   0.0000   0.7071
  -0.5774   0.7071   0.7071


D =                                       Column 1 of V is the eigenvector corresponding
  -1.0000        0        0               to eigenvalue D(1,1) = -1; column 2 of V is for
        0   1.0000        0               D(2,2) = 1; and column 3 of V is for D(3,3) = 2.
        0        0   2.0000
```

PROGRAM: **Write a program to determine the eigen vector and eigen values of A = [1 2 3; 4 5 6; 7 8 9]**

```
>> A = [ 1  2  3;  4  5  6; 7  8  9]
>> eig(A)
>> [V D]=eig(A)
```

RESULT :

```
eig(A) =

   16.1168

   -1.1168

   -0.0000

V =

   -0.2320   -0.7858    0.4082

   -0.5253   -0.0868   -0.8165

   -0.8187    0.6123    0.4082

D =

   16.1168         0         0

         0   -1.1168         0

         0         0   -0.0000
```

CONCLUSION: Hence we have written program to determine eigenvectors and eigen-values of a square matrix and the result has been found out.

PRE-EXPERIMENTAL QUESTIONS

Q1. What do you mean by eigen-values of a square matrix?

Q2. What do you mean by eigen-vectors of a square matrix?

Q3. How do you solve eigen-values and eigen-vectors of a square matrix manually?

POST-EXPERIMENTAL QUESTIONS

Q4.How do you solve eigen-values and eigen-vectors of a square matrix in MATLAB?

EXPERIMENT NO. 6

OBJECTIVE: TO DETERMINE SOLUTION OF DIFFERENCE EQUATIONS

APPARATUS/ SOFTWARE REQUIRED:

SR.NO	APPARATUS/SOFTWARE	SPECIFICATION	QUANTITY

THEORY:

DIFFERENTIAL EQUATIONS:

A **differential equation** is a mathematical equation for an unknown function of one or several variables that relates the values of the function itself and its derivatives of various orders. Differential equations play a prominent role in engineering, physics, economics, and other disciplines.

GIVEN NUMERICAL:

The function 'dsolve' computes symbolic solutions to ordinary differential equations. The equations are specified by symbolic expressions containing the letter 'D' to denote differentiation. The symbols D2, D3,….DN, correspond to the second, third,….,Nth derivative, respectively. Thus, D2y is d^2/dt^2. The dependent variables are those preceded by D and the default independent variable is t. Note that names of symbolic variables should not contain D. The key issues in this example are the order of the equation and the initial conditions.

To solve the ordinary differential equation, simply type:

y = dsolve('D3y=y', 'y(0)=1', 'Dy(0)=-1', 'D2y(0)=pi', 'x')

where D3y represents d^3y/dx^3 and D2y(0) represents d^2y/dx^2 at x = 0.

Examples

(i) dy/dt = -ay

(ii) dy/dt = -ay and y(0) =1

(iii) d^2y/dt^2 = -$-a^2$ y and y(0)= 1, dy/dt(Π/a)=0

(iv) dy/dx = (xy - y^2)/ x^2

(v) dy/dx = tan(y/x) +y/x

COMMANDS:

(i)

>> y=dsolve('Dy=-a*y') %write ODE in inverted comma

(ii)

>> y=dsolve('Dy=-a*y', 'y(0)=1') %write ODE in inverted comma followed by initial condition, separated by comma

(iii)

>> y=dsolve('D2y=-a^2*y', 'y(0)=1, Dy(pi/a)=0')

%write ODE in inverted comma followed by initial condition, separated by comma

Note: In all the above cases, the independent variable is 't' by default.

(iv)

>> y = dsolve ('Dy=(x*y –y^2)/ x^2', 'x')

% define independent variable as 'x'

(v)

```
>> y = dsolve ('Dy=tan(y/x) + y/x)', 'x')
```

RESULTS:

(i)

```
>> y=dsolve('Dy=-a*y')
y =
C1/exp(a*t)
```

(ii)

```
>>y=dsolve('Dy=-a*y', 'y(0)=1')
y=
exp(-a*t)
```

(iii)

```
>>y=dsolve('D2y=-a^2*y', 'y(0)=1, Dy(pi/a)=0')
y=
cos(a*t)
```

(iv)

```
>> y = dsolve ('Dy=(x*y –y^2)/ x^2', 'x')
y=
-x/(C12 - log(x))
```

(v)

```
>> y = dsolve ('Dy=tan(y/x) + (y/x)', 'x')
y=
asin(x*C1)*x
```

PRE-EXPERIMENTAL QUESTIONS

Q1.What is a **differential equation** ?

Q2. Give examples of **differential equations.**

POST-EXPERIMENTAL QUESTIONS

Q3. What command is used to solve **differential equations** using MATLAB? Explain with examples.

Q4. Write programs to solve :

i) dy/dx = -($y^2 - x^2/2xy$)

ii) dy/dx = (2y - x)/(2x -y)

iii) dy/dx = ytanx - $y^2 \sec x$

iv) dy/dx = 3 $e^x y^3$ - y

v) $\frac{d^2y}{dt^2} - \frac{8dy}{dx} + 15y = 0$

EXPERIMENT NO. 7

OBJECTIVE: SOLUTION OF DIFFERENTIAL EQUATION USING EULER METHOD

APPARATUS/ SOFTWARE REQUIRED:

SR.NO	APPARATUS/SOFTWARE	SPECIFICATION	QUANTITY

THEORY:

Let's consider a first-order differential equation:

$$y'(t) + ay(t) = r \qquad with\ y(0) = y_0 \tag{1}$$

It has the following form of analytical solution

$$y(t) = (y_0 - \frac{r}{a})e^{-at} + \frac{r}{a} \tag{2}$$

Which can be obtained by using a conventional method or the Laplace transform technique.

First of all, we have to replace the derivative $y'(t) = \frac{dy}{dt}$ in the differential equation by a numerical derivative , where the step size h is determined based on the accuracy requirements and the computation time constraints. Euler's method approximates as

$$\frac{y(t+h) - y(t)}{h} + ay(t) = r$$

$$y(t+h) = (1-ah)y(t) + hr \quad with\ y(0)$$

And solve this difference equation step-by-step with increasing t by h each time from t = 0.

$$y(h) = (1-ah)y(0) + hr = (1-ah)y_0 + hr$$

$$y(2h) = (1-ah)y(h) + hr = (1-ah)^2\, y_0 + (1-ah)hr + hr$$

$$y(3h) = (1-ah)y(2h) + hr = (1-ah)^3\, y_0 + \sum_{m=0}^{2} (1-ah)^m\, hr$$

This is a numeric sequence $\{y(kh)\}$ which we call a numerical solution of Eqn. (1).

To be specific, let the parameters and the initial value of Eqn.(1) be a = 1, r = 1, and $y_0 = 0$. Then the analytical solution of eqn.(2) becomes

$$y(t) = 1 - e^{-at}$$

PROGRAM:

% **Euler method to solve a 1st order differential equation of $y(t) = 1 - e^{-at}$** %

a = 1; r = 1; $y\_0 = 0$; tf = 2;

t = [0:0.01:tf];

yt = 1-exp(-a *t);

plot (t, yt, 'k'), hold on

klasts = [8 4 2]; hs = tf. /klasts;

```
y(1) = y_0;
for int = 1: 3
klast = klasts (int) ;  h = hs(int) ;  y(1) = y_0;
for k = 1: klast
    y(k+1) = (1-a*h)*y(k) + h* r ;
     plot ([k-1 k]*h, [y(k) y(k+1)], 'b' , k* h , y(k+1), 'r')
    if  k< 4, pause ; end
    end
 end
```

RESULT & OBSERVATIONS:

2.GIVEN NUMERICAL:

dy/dx = (y - x)/(y + x)

COMMANDS:

>> b = 3; a = 0;m = 4; x = 0; y = 1;

>> h = (b-a)/m;

>>x = a:h:b;

>> for j = 1:m;

>> y(j+1)=y(j)+h*((y(j)-x(j))/ (y(j)+x(j)));

end

>>E = [x' y']

RESULTS:

E=[x' y']

E = 0   1.0000

0.7500   1.7500

1.5000   2.0500

2.2500   2.1662

3.0000   2.1520

CONCLUSION:

EXPERIMENT NO. 8

OBJECTIVE: SOLUTION OF DIFFERENTIAL EQUATIONS USING 4th ORDER RUNGE-KUTTA METHOD

APPARATUS/ SOFTWARE REQUIRED:

SR.NO	APPARATUS/SOFTWARE	SPECIFICATION	QUANTITY

THEORY:

The fourth-order Runge -Kutta (RK4) method having a truncation error of O(h^4) is one of the most widely used methods for solving differential equations. Its algorithm is described below:

$$y_{k+1} = y_k + \frac{h}{6}(f_{k1} + 2f_{k2} + 2f_{k3} + f_{k4}) \quad (1)$$

Where

$$f_{k1} = f(t_k,\ y_k) \quad (2)$$

$$f_{k2} = f\left(\ t_k + \frac{h}{2}, y_k + f_{k1}\frac{h}{2}\ \right) \quad (3)$$

$$f_{k3} = f\left(\ t_k + \frac{h}{2}, y_k + f_{k2}\frac{h}{2}\ \right) \quad (4)$$

$$f_{k4} = f(\ t_k + h,\ y_k + f_{k3}h\ ) \quad (5)$$

PROGRAM:

```
function  [t, y] = ode_RK4(f , tspan , y0, N, varargin)
  % Runge- Kutta method to solve vector differential equation  y'(t) = f(t, y(t))
  % for tspan = [t0, tf] and with the initial value y0 and N time steps
if nargin <  4  |       N<= 0, N = 100; end
if nargin  <  3, y0 = 0; end
y(1, :) = y0(:)' ;                                          %  make it a row vector
h = (tspan(2) – tspan(1)) / N ;     t = tspan(1) + [0: N]' * h ;
for k = 1: N
     f1 = h* feval (f , t(k), y(k, :) ,  varargin{:});  f1 = f1(:)';
     f2 = h* feval (f , t(k) + h/2 , y(k, :) + f1/2 ,  varargin{:});  f2 = f2(:)';
     f3 = h* feval (f , t(k) + h/2 , y(k, :) + f2/2 ,  varargin{:});  f3 = f3(:)';
     f4 = h* feval (f , t(k) + h , y(k, :) + f3 ,  varargin{:});  f4 = f4(:)';
     y(k + 1 , :) = y(k, :) + (f1 + 2*(f2 +f3) + f4 )/6 ;
end
```

RESULT & DISCUSSIONS:

CONCLUSION:

EXPERIMENT NO. 9

OBJECTIVE: DETERMINATION OF ROOTS OF A POLYNOMIAL

APPARATUS/ SOFTWARE REQUIRED:

SR.NO	APPARATUS/SOFTWARE	SPECIFICATION	QUANTITY

THEORY:

POLYNOMIAL:

Polynomials are functions that have the form

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n$$

The coefficients $a_1, a_2, \ldots, a_n$ are often real numbers and n which is a nonnegative integer, is the *degree* or *order* of the polynomial.

IN MATLAB:

Polynomials are described by using a row vector of the coefficients of the polynomial beginning with the highest power of *x* and inserting zeros for "missing" terms:

$$f(x) = 9x^3 - 5x^2 + 3x + 7$$

$$g(x) = 6x^2 - x + 2$$

```
f= [9   -5   3   7]

g= [6   -1   2]
```

ROOTS OF A POLYNOMIAL:

Roots of a polynomial

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n$$

are the values of x for which $f(x) = 0$. For example, the roots of $f(x) = x^2 - 3x + 2$ are -1 and -2. There are *n* roots of a polynomial with degree *n*.

The command "***roots"*** determines the roots of a polynomial. The usage of the function is:

r= roots (p)

where *r* is a column vector with the roots and p is a row vector with the coefficients of the polynomial.

PROGRAM:

Find the roots of the polynomial

$$f(x) = 3x^6 + 15x^5 - 10x^3 + 4x$$

>> p = [3 15 0 -10 0 4 0]

>> roots(p)

RESULT:

0

-4.8613

-0.6925 + 0.3093i

-0.6925 - 0.3093i

0.6232 + 0.2975i

0.6232 - 0.2975i

CONCLUSION:

PRE-EXPERIMENTAL QUESTIONS:

1. **What is a polynomial?**

2. **What do you mean by roots of a polynomial?**

POST-EXPERIMENTAL QUESTIONS:

1. **Which command is used to find the roots of a polynomial?**
2. **Solve the following equations:**

   i) $\boldsymbol{x^4 - 16x^3 + 86x^2 - 176x + 105 = 0}$

   ii) $\boldsymbol{x^4 - 3x^2 + 42x - 40 = 0}$

   iii) $\boldsymbol{x^4 + 12x - 5 = 0}$

EXPERIMENT NO. 10

OBJECTIVE: DETERMINATION OF POLYNOMIAL USING METHOD OF LEAST SQUARE CURVE FITTING

APPARATUS/ SOFTWARE REQUIRED:

SR.NO	APPARATUS/SOFTWARE	SPECIFICATION	QUANTITY

THEORY:

Curve fitting is a technique of finding an algebraic relationship that "best" (in least squares sense) fits a given set of data. Unfortunately, there is no magical function (in MATLAB or otherwise) that can give the relationship if we simply supply the data. We have to have an idea of what kind of relationship might exist between the input data and the output data. However, if we do not have the firm idea but have data that we trust, MATLAB can help us in exploring the best possible fit.

MATLAB includes ***Basic Fitting*** in its Figure window's Tools menu that lets us fit a polynomial curve (upto 10^{th} order) to the data on the fly. It also gives us options of displaying the residual at the data points and computing the norm of the residuals. This can help in comparing different fits and then selecting the one that best fits.

1. **GIVEN NUMERICAL:**

   Fit a straight line y = a+ bx to the following data:

x: 1 2 3 4 5

y: 14 27 40 55 68

COMMANDS:

i) >>x = [1 2 3 4 5]; %write the elements with spacing

>> y = [14 27 40 55 68];

>> polyfit(x,y,1) %fits the data to linear equation

ii) >> x=[0 1 2 3 4];

>> y=[1 5 10 22 38];

>> polyfit(x,y,2) %fits the data to quadratic equation

>>plot(x, y)

>>title ('curve fitting')

RESULTS:

i) 13.6000 -0.0000

The constants a and b are 0 and 13.6 respectively.

Therefore, equation of the straight line is:

y=13.6x

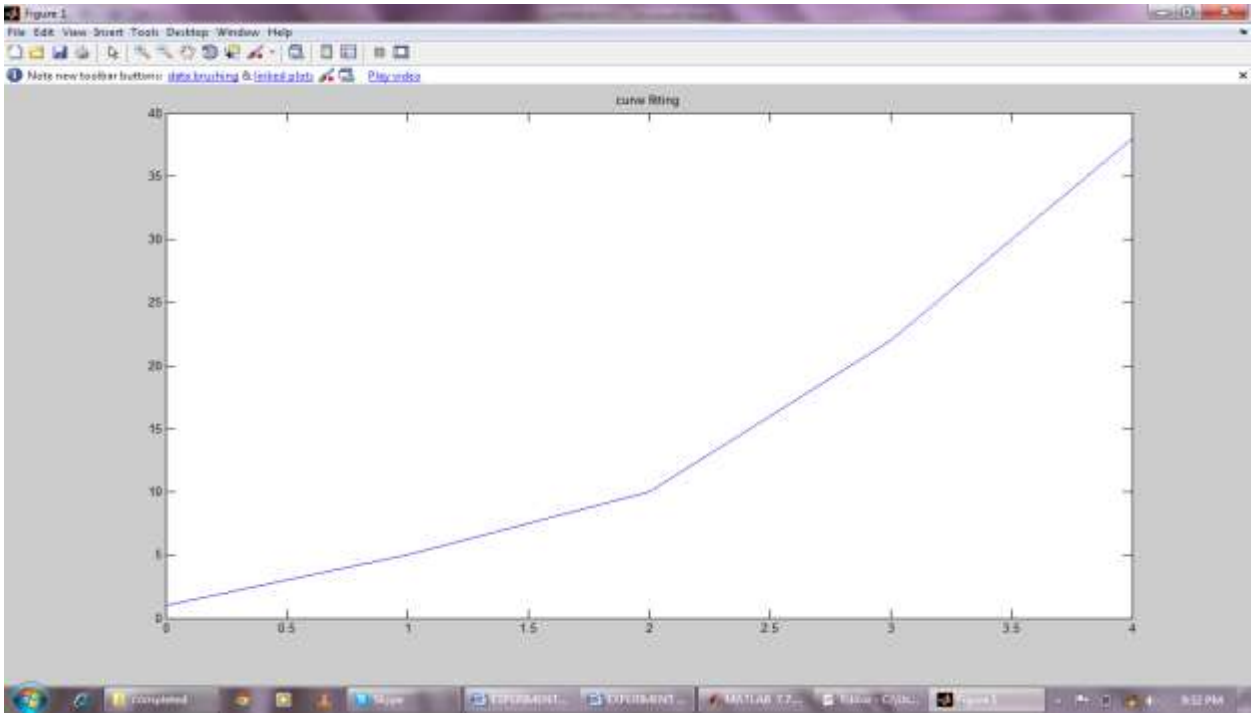
ii) 2.2143 0.2429 1.4286

The constants of quadratic equation $y = a+bx+cx^2$ are a = 2.2143, b = 0.2429 and c= 1.4286

Hence quadratic equation is:

$y= 1.4286x^2 + 0.2429x + 2.2143$

The graph of the quadratic equation is shown in fig.below.



2. Using MATLAB curve fitting toolbox:

   Proceed as mentioned below:

   i) Write down the given data on command window for two variables.

   ii) Plot the curve for two variables

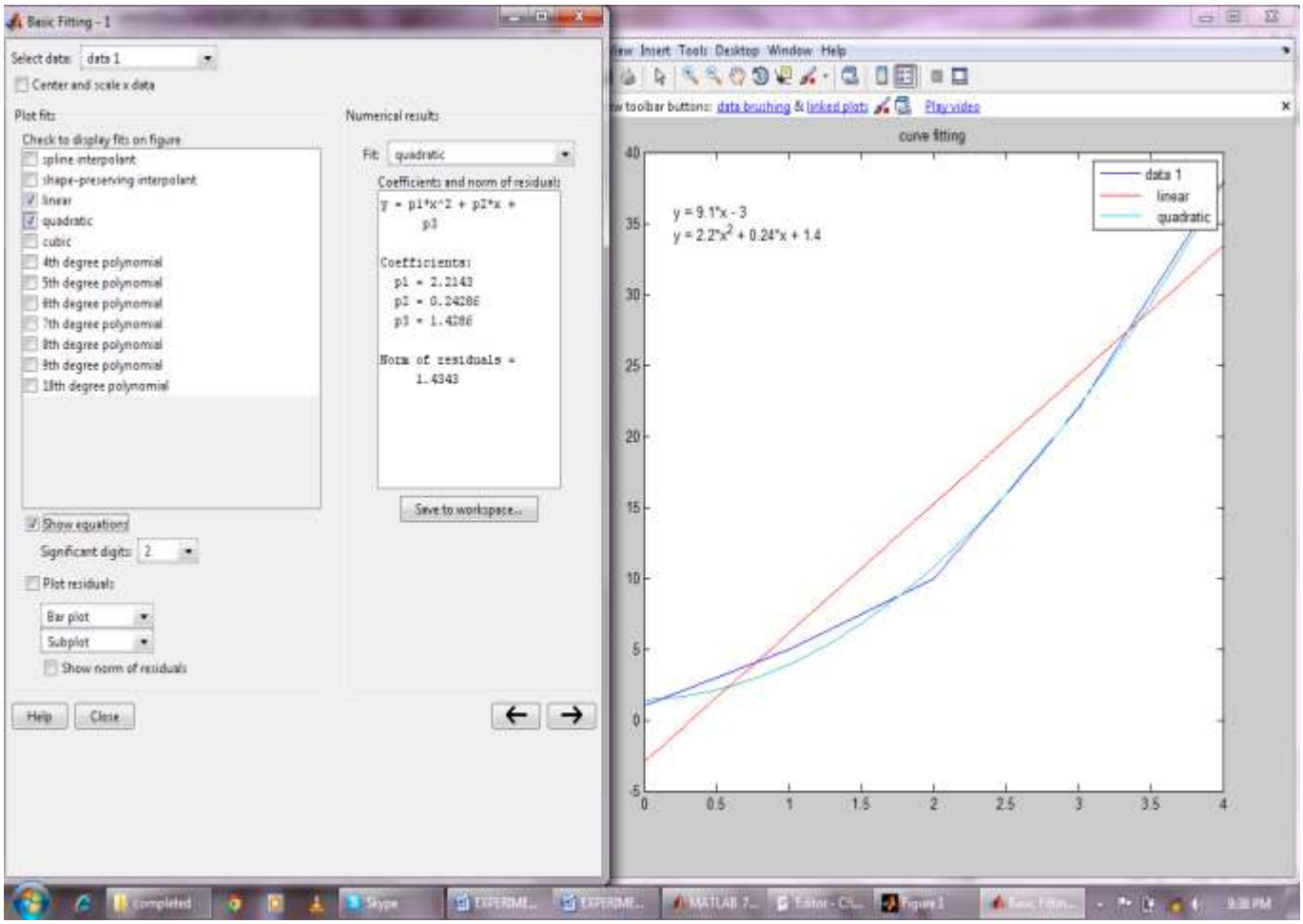
   iii) Go to figure window

   iv) Go to tools

   v) Go to basic fitting

   vi) Select linear or quadratic function or higher order, to which the given data is to be fitted.

   vii) You may also select 'show equation'. On selection, the equation appears on the figure window.


Basic Fitting - 1
Select data: data 1
Center and scale x data
Plot fits
Check to display fits on figure
spline interpolant
shape-preserving interpolant
linear
quadratic
cubic
4th degree polynomial
5th degree polynomial
6th degree polynomial
7th degree polynomial
8th degree polynomial
9th degree polynomial
10th degree polynomial
Show equations
Significant digits: 2
Plot residuals
Bar plot
Subplot
Show norm of residuals
Help
Close
Numerical results
Fit: quadratic
Coefficients and norm of residuals
y = p1*x^2 + p2*x + p3
Coefficients:
p1 = 2.2143
p2 = 0.2428
p3 = 1.4286
Norm of residuals = 1.4343
Save to workspace...
curve fitting
data 1
linear
quadratic
y = 9.1*x - 3
y = 2.2*x^2 + 0.24*x + 1.4

CONCLUSION:

PRE-EXPERIMENTAL QUESTIONS

Q1. What is Curve fitting?

POST-EXPERIMENTAL QUESTIONS

Q3. By the method of least squares, find the straight line that fits the following data:

x: 1	2	3	4	5
y:14	27	40	55	68

Q4. Fit a parabola $y=ax^2+bx+c$ in least square sense to the data

x: 10	12	15	23	20
y:14	17	23	25	21

EXPERIMENT NO. 12

OBJECTIVE: DETERMINATION OF TIME RESPONSE OF AN R-L-C CIRCUIT

APPARATUS/ SOFTWARE REQUIRED:

SR.NO	APPARATUS/SOFTWARE	SPECIFICATION

GIVEN NUMERICAL

An R-L-C circuit has R = 180 ohms, C = 1/280 farads, L = 20 henries and an applied voltage E(t) = 10 sin t. Assuming that no charge is present but an initial current of I ampere is flowing at t = 0 when the voltage is first applied, find q and i =dq/dt at any time t. q is diven by the differential equation.

$L\, d^2q/dt^2 + R\frac{dq}{dt} + \frac{q}{c} = E(t)$

COMMANDS:

>> syms q t % declare q, t as symbolic

>> q = dsolve('20*D2q +180*Dq +280*q = 10*sin(t)', 'q(0)=0')

>> simplify(q) %simplify the result

>> pretty(q) %print in readable form

>> i=diff(q) %current(i) is differentiation of 'q' wrt 't'

>>pretty(i)

RESULTS:

q=

exp(-2*t)*C2+exp(-7*t)*(-C2+9/500)-9/500*cos(t)+13/500*sin(t)

>> **pretty(q)**

$$\exp(-2\,t)\,C2 + \exp(-7\,t)\left(-C2+\frac{9}{500}\right) - \frac{9}{500}\cos(t) + \frac{13}{500}\sin(t)$$

>> **i=diff(q)**

i=

-2*exp(-2*t)*C2 – 7*exp(-7*t)*(-C2 +9/500)+ 9/500*sin(t)+13/500 *cos(t)

>> **pretty(i)**

$$-2\exp(-2t)\,C2-7\exp(-7\,t)(-C2+9/500)+9/500\sin(t)+\frac{13}{500}\cos(t)$$

CONCLUSION:

PRE-EXPERIMENTAL QUESTIONS:

Q1. What is an R-L-C circuit?

Q2. What do you mean by time response of a circuit?

POST-EXPERIMENTAL QUESTIONS:

Q3. For an electric circuit with circuit constants L, C, R the charge q on the plate of the condenser is given by:

Ld^2q/dt^2+ Rdq/dt + q/C = 0. Find q at any time t.

Q4.A 10.3 farad capacitor is connected in series with 0.05 henry inductor and 10 ohm resistor. Initially the current in the circuit is zero and the charge on the capacitor is also zero. If the emf is 50 sin 200t. Find the charge after a period t seconds of closing the circuit.