MGM

JAWAHARLAL NEHRU ENGINEERING COLLEGE
AURANGABAD

S.Y. B.TECH

ELECTRICAL ENGINEERING (ELECTRONICS & POWER)

NUMERICAL METHODS & PROGRAMMING

LAB MANUAL

- PROF. MADHURESH MADHAV SONTAKKE -

DEPARTMENT OF

ELECTRICAL ELECTRONICS & POWER ENGINEERING

LABORATORY MANNUAL CONTENTS

This manual is intended for the second year students of B.tech Electrical Engineering (Electronics & Power) for the LAB of Numerical Methods & Programming. This manual typically contains practical/Lab Sessions related to Numerical Methods & Programming using MATLAB covering various aspects related the subject to enhance understanding.

In this manual we have made the efforts to cover various methods & programs in MATLAB for various tedious numerical methods & computational techniques.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions.

–Prof. Madhuresh Madhav Sontakke

MAHATMA GANDHI MISSION`S

JAWAHARLAL NEHRU ENGINEERING COLLEGE

AURANGABAD.

Department of Electrical Electronics and Power

Vision of JNEC

To create self-reliant, continuous learner and competent technocrats imbued with human values.

Mission of JNEC

1. Imparting quality technical education to the students through participative teaching – learning process.
2. Developing competence amongst the students through academic learning and practical experimentation.
3. Inculcating social mindset and human values amongst the students.

JNEC Environmental Policy

1. The environmental control during its activities, product and services.
2. That applicable , legal and statutory requirements are met according to environmental needs.
3. To reduce waste generation and resource depletion.
4. To increase awareness of environmental responsibility amongst its students and staff.
5. For continual improvement and prevention of pollution.

JNEC Quality Policy:

Institute is committed to:

- To provide technical education as per guidelines of competent authority.
- To continually improve quality management system by providing additional resources required. Initiating corrective & preventive action & conducting management review meeting at periodical intervals.
- To satisfy needs & expectations of students, parents, society at large.

EEP Department

VISION:

- To Develop competent Electrical Engineers with human values.

MISSION:

- To Provide Quality Technical Education To The Students Through Effective Teaching-learning Process.
- To Develop Student's Competency Through Academic Learning, Practicals And Skill Development Programs.
- To Encourage Students For Social Activities & Develop Professional Attitude Along With Ethical Values.

Computer Lab DO's and DON'T

Do's

1. Know the location of the fire extinguisher and the first aid box and how to use them in case of an emergency.

2. Read and understand how to carry out an activity thoroughly before coming to the laboratory.

3. Report fires or accidents to your lecturer/laboratory assistant immediately.

4. Report any broken plugs or exposed electrical wires to your lecturer/laboratory assistant immediately.

Don'ts

1. Do not eat or drink in the laboratory.

2. Avoid stepping on electrical wires or any other computer cables.

3. Do not open the system unit casing or monitor casing particularly when the power is turned on. Some internal components hold high electric voltages which can be fatal.

4. Do not insert metal objects such as clips, pins and needles into the computer casings. They may cause fire.

5. Do not remove anything from the computer laboratory without permission.

6. Do not touch, connect or disconnect any plug or cable without your lecturer/laboratory technician's permission. 7. Do not misbehave in the computer laboratory.

List of Experiments

Sr no	Name of Experiment
1	Study of Introduction to MATLAB
2	Study of basic matrix operations
3	Program for scan conversion of a straight line
4	Program for scan conversion of a circle
5	Program for scan conversion of an ellipse
6	Program for scan conversion of a rectangle
8	Program for scan conversion of an arc
9	Program To solve linear equation
10	Program for finding roots of f(x)=0 by newton raphsonm method
11	Program for finding roots of f(x)=0 by bisection method
12	Program for solving numerical integration by simpson's 1/3 rule
13	Program for solving ordinary differential equation by runge kutta method

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.

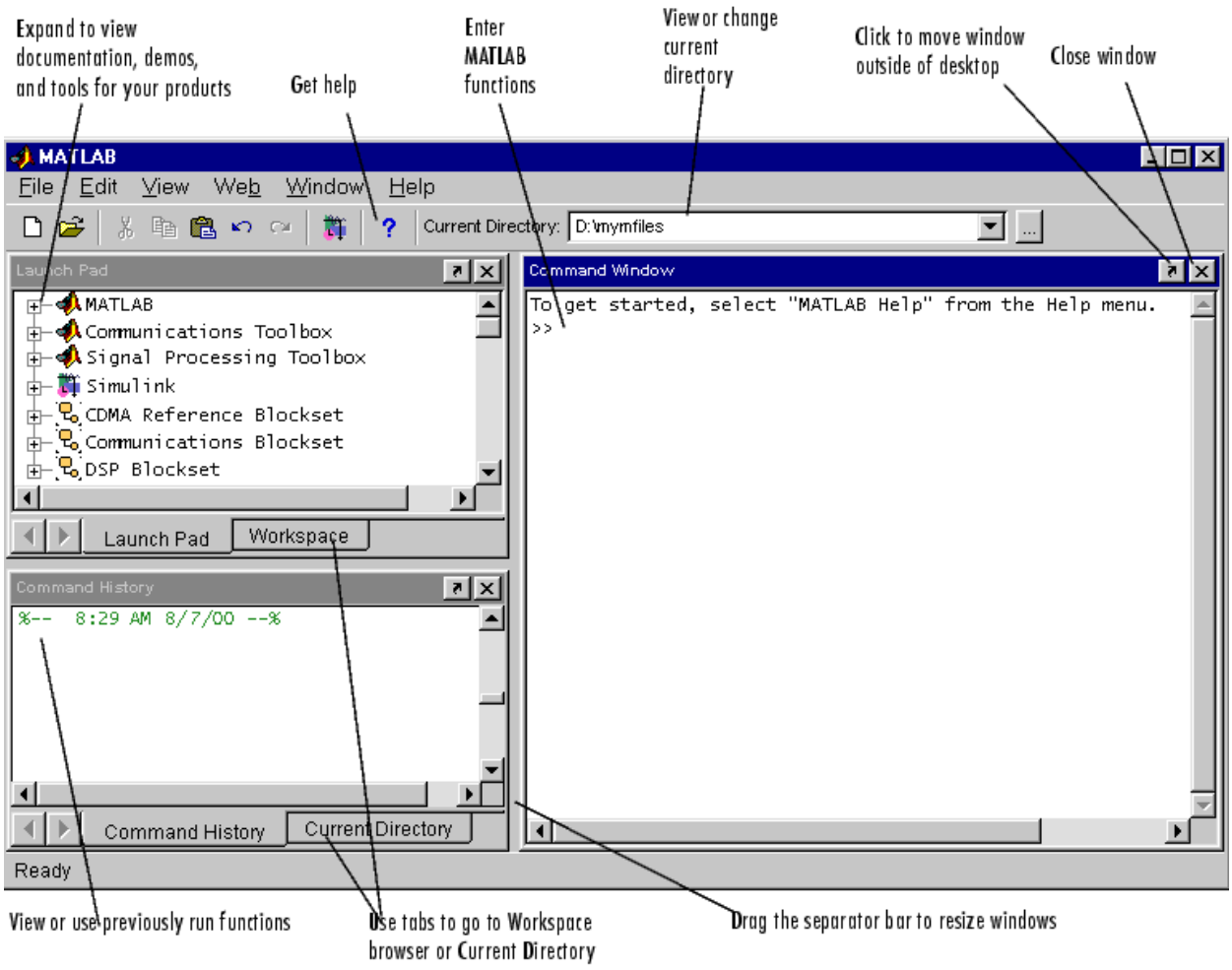
2. BASICS:

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 :

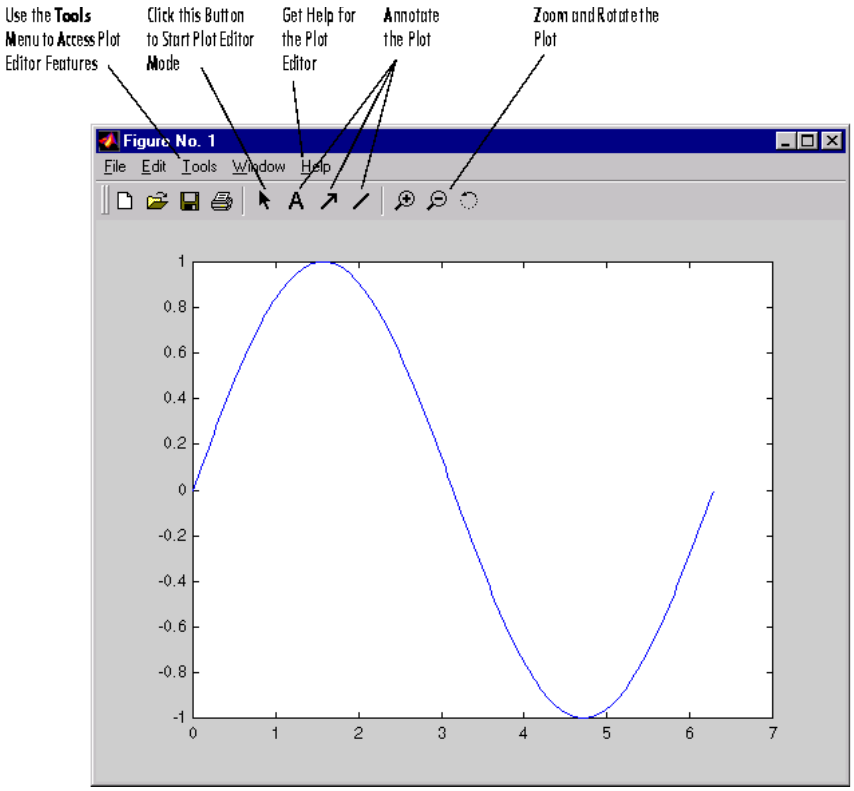
- **COMMAND WINDOW:** This is the main window , where command are written. It is characterized by MATLAB command prompt (>>). The results also appear on this window(except figures, which appear on figure window) which command is written. Commands cannot be edited in this window.

- **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.



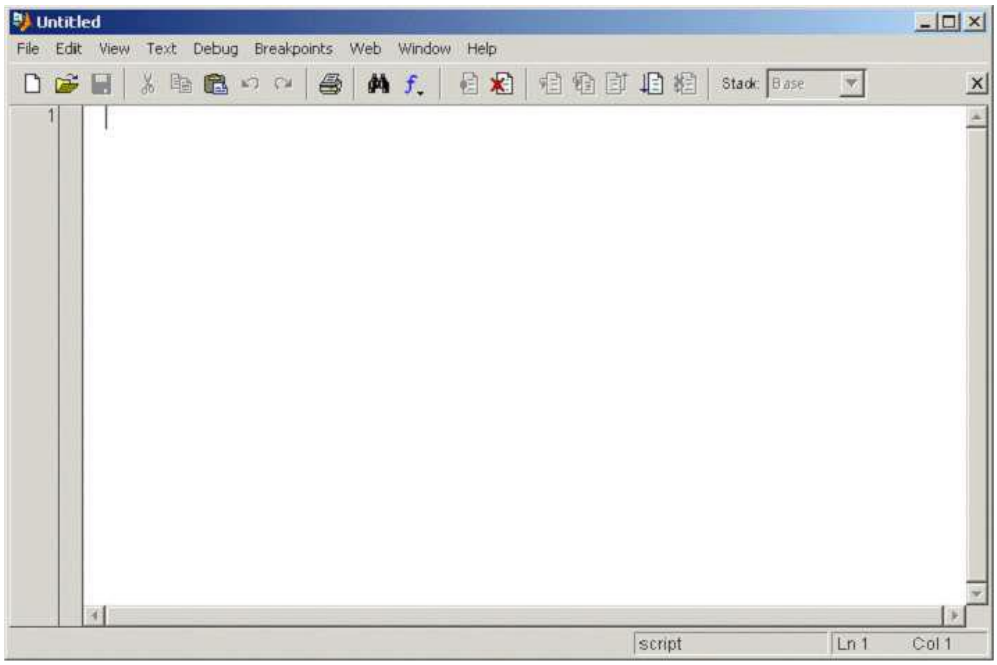
2. FIGURE WINDOW:

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 programs written on the file are first saved and then run to get the results. To save and run the program, 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..

Conclusion :

Experiment no 2

OBJECTIVE: STUDY OF BASIC ARRAY & MATRIX OPERATIONS.

Theory :

Arrays and Vectors

An array is a collection of data objects of the same type, typically stored sequentially in memory.

- An array is an elementary data structure.
- Almost all programming languages provide support for arrays.
- Matlab is a language that has been particularly specialized to support arrays (and subsequently matrices).
- An array is the obvious way to represent a vector.
- A 3D vector with coordinates [1, 2.5, 5] would be represented as an array of 3 doubles arranged sequentially in memory.

Declaring (Constructing) Arrays

- In Matlab, we can construct an array and associate it with an identifier very easily.

For example: >> a = [1, 2.5, 5] a = 1.0000 2.5000 5.0000

- The commas are optional and can be omitted:

  >> a = [1 2.5 5] a = 1.0000 2.5000 5.0000

- The square brackets indicate to Matlab that the contents represent an array:

>> whos Name Size Bytes Class a 1x3 24 double array Grand total is 3 elements using 24 bytes

Operations on Arrays

- This process of constructing an array involves a segment of memory being allocated and associated with the variable name, and the elements of memory being set to the specified values.
- In most programming languages you would have to declare an array and assign the values one at a time. In Matlab this is automatic.
- You can then perform arithmetic on arrays as simply as you can with scalars. For example:

  >> b = a*2 b = 2 5 10

Working with Array Elements

- You can extract individual values from an array by specifying the index within the array using round brackets. For example:

  >>c = b(1) % c is set to the value of the first element of b c = 2
- You can also assign new values to individual elements of an array. For example:

  mple: >> b(3) = 6 % Set the value of the 3rd element of b to 6

  b = 2 5 6
- In Matlab, the index of the first element of an array is always 1.
- Note: this differs from languages such a C or Java where the index of the first element is always 0.

Matrices

- An array is a collection of data objects of the same type.
- The data objects of the array can themselves be arrays.
- A matrix is typically represented by an array of arrays, or a 2D array. Matlab supports matrices in the same way that it supports vectors.
- Matlab uses the semi-colon (;) operator to distinguish between the different rows of a matrix. For example:

```
>> a = [1 2 3; 4 5 6]          % The ; separates the
                               % individual 1D arrays.
    a =
    1     2     3
    4     5     6
```

Everything in Matlab is a Matrix

- Matlab also allows rows to be entered on different lines.
- Once an array is started by a square bracket ([), Matlab assumes that a new line means a new row of the matrix. For example:

```
>> a = [1     2     3                % A matrix consisting of two rows
        4     5     6 ];
```

- As far as Matlab is concerned, everything is a matrix!
- A vector is a 1xN (or Nx1) matrix; a scalar is a 1x1 matrix.

Matrix and vector operators

Experiment no 3

SCAN CONVERTING LINES

Objective : TO study scan conversion of Line

Theory :

Description: Given the specification for a straight line, find the collection of addressable pixels which most closely approximates this line.

- Straight lines should appear straight.
- Lines should start and end accurately, matching endpoints with connecting lines.
- Lines should have constant brightness.
- Lines should be drawn as rapidly as possible.

Problems Associate with the Drawing Straight line

- How do we determine which pixels to illuminate to satisfy the above goals?
- Vertical, horizontal, and lines with slope = +/- 1, are easy to draw.
- Others create problems: stair-casing/ jaggies/aliasing. Quality of the line drawn depends on the location of the pixels and their brightness

In short It is difficult to determine whether a pixel belongs to an object.
The general goal is to minimize the stair-step effect (jaggies), have a uniform line density, and require the minimum drawing time. Remember, we must always round or truncate to integers since pixels are at integer positions.

Following Algorithms are used for drawing Line.

- Floating Point Algorithms
- Digital Differential Analyzer (DDA) method
- Bresenham (Decision Variable) method
- Point and Line Drawing Commands
- Demonstration Program for scan converting lines

1. Floating Point Algorithms

The equation for a straight line: y = m * x + b, where m is the slope (m = dy/dx) and b is the y intercept. So for a given x interval (dx) we can compute dy = m * dx, but this is a brute force real number computation and is very slow.

Alternative Method - Parametric Equations

For a line segment x1, y1 -> x2, y2:

x = x1 + t*dx with dx = x2 - x1 and 0.0 <= t <= 1.0

So start with t = 0 and increment to t = 1.0, e.g. t= .05, .10,.15, etc. This is still too slow but parametric equations are used for clipping, curve fitting, and ray tracing.

2. Digital Differential Analyzer (DDA) Method

The basis of the DDA method is to take unit steps along one coordinate and compute the corresponding values along the other coordinate. The unit steps are always along the coordinate of greatest change, e.g. if dx = 10 and dy = 5, then we would take unit steps along x and compute the steps along y.

Let us assume a line of positive slope and draw from x1, y1 to x2, y2.

{slope = m = dy/dx}

```
if m <= 1.0 then

  let x_step = 1 {dx = 1, dy = 0 or 1}
else {m > 1.0}
  let y_step = 1 {dy = 1, x_step = 0 or 1 }
```

Remember: always step along the longest delta.

{m <= 1.0} for x_step = 1, dy = m = yi+1 - yi -> yi+1 = yi + m

{m > 1} for y_step = 1 m = 1/dx => dx = 1/m => xi+1 = xi + 1/m

Actual MATLAB PROGRAM

```
function draw(x0,y0,x1,y1)
clc
clear
point = input('Give coord[ x0 y0 x1 y1]: ');
x0 = point(1);y0 = point(2); x1 = point(3);y1=point(4);
dx = abs(x1-x0);
dy = abs(y1-y0);
sx = sign(x1-x0);
sy = sign(y1-y0);
if (dy > dx)
    step = dy;
else
    step = dx;
end
x(1) = x0; y(1) = y0; j = 1;
for i= 0:1:step
    if (x1 == x)&(y1 == y)
        break;
    end
    j = j+1;
    x(j) = x(j-1) + (dx/step)*sx;
    y(j) = y(j-1) + (dy/step)*sy;
end
plot(round(x),round(y),'rs');
grid on ,hold on

plot(x,y,'b^');
axis([point(1)-2 point(3)+2 point(2)-2 point(4)+2]);
legend('DDA Points','Actual points',2);
title('Digital Differential Line Drawing Algorithm')
```

Experiment no 4

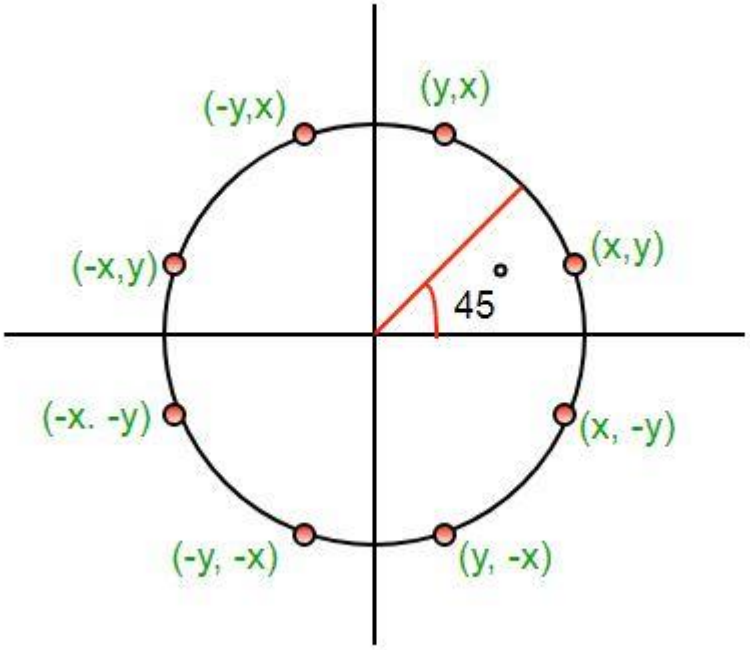
SCAN CONVERTING CIRCLE

Objective : TO study scan conversion of Circle

Theory :

We need to plot the perimeter points of a circle whose center co-ordinates and radius are given using the Mid-Point Circle Drawing Algorithm.

We use the above algorithm to calculate all the perimeter points of the circle in the first octant and then print them along with their mirror points in the other octants. This will work only because a circle is symmetric about it's centre.



The algorithm is very similar to the Mid-Point Line Generation Algorithm. Here, only the boundary condition is different.

For any given pixel (x, y), the next pixel to be plotted is either (x, y+1) or (x-1, y+1). This can be decided by following the steps below.

1. Find the mid-point **p** of the two possible pixels i.e (x-0.5, y+1)
2. If **p** lies inside or on the circle perimeter, we plot the pixel (x, y+1), otherwise if it's outside we plot the pixel (x-1, y+1)

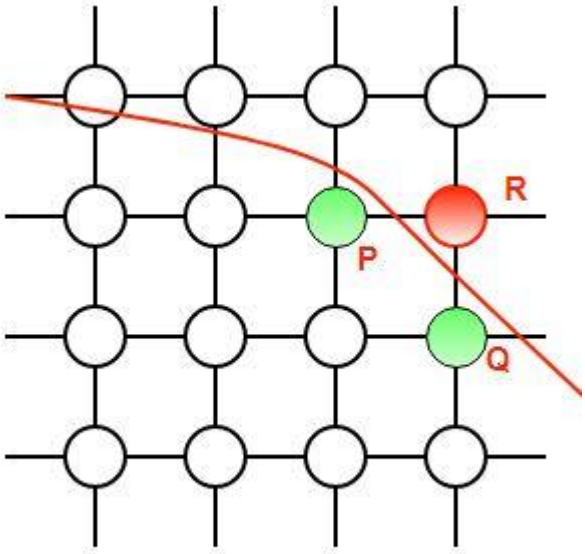
Boundary Condition : Whether the mid-point lies inside or outside the circle can be decided by using the formula:-

Given a circle centered at (0,0) and radius r and a point p(x,y)
F(p) = x2 + y2 – r2

if F(p)<0, the point is inside the circle

F(p)=0, the point is on the perimeter

F(p)>0, the point is outside the circle



MATLAB PROGRAM

```
function h = circle(x,y,r)
Z=input('Enter the coordinates as [x,y] : ');
x = Z(1);
y = Z(2);
r = input('Enter the radius : ');
th = 0:pi/50:2*pi;
xunit = r * cos(th) + x;
yunit = r * sin(th) + y;
plot(xunit, yunit);
hold off
```

Experiment No 5

Scan convergence of Ellipse

Objective : Drawing ellipse using MATLAB script.

Theory : In mathematics, an ellipse is a curve in a plane surrounding two focal points such that the sum of the distances to the two focal points is constant for every point on the curve. As such, it is a generalization of a circle, which is a special type of an ellipse having both focal points at the same location. The elongation of an ellipse is represented by its eccentricity, which for an ellipse can be any number from 0 (the limiting case of a circle) to arbitrarily close to but less than 1.

Ellipses are the closed type of conic section: a plane curve resulting from the intersection of a cone by a plane (see figure to the right). Ellipses have many similarities with the other two forms of conic sections: parabolas and hyperbolas, both of which are open and unbounded. The cross section of a cylinder is an ellipse, unless the section is parallel to the axis of the cylinder.

Analytically, an ellipse may also be defined as the set of points such that the ratio of the distance of each point on the curve from a given point (called a focus or focal point) to the distance from that same point on the curve to a given line (called the directrix) is a constant. This ratio is the above-mentioned eccentricity of the ellipse

Definition of an ellipse as locus of points

An ellipse can be defined geometrically as a set of points (locus of points) in the Euclidean plane:

An ellipse is a curve that is the locus of all points in the plane the sum of whose distances r_1 and r_2 from two fixed points F_1 and F_2 (the foci) separated by a distance of $2c$ is a given positive constant $2a$.This results in the two-center bipolar coordinate equation

$$r_1 + r_2 = 2a,$$

where a is the semimajor axis and the origin of the coordinate system is at one of the foci. The corresponding parameter b is known as the semiminor axis.

The ellipse is a conic section and a Lissajous curve.

An ellipse can be specified in the Wolfram Language using Circle[{x, y}, {a, b}].

If the endpoints of a segment are moved along two intersecting lines, a fixed point on the segment (or on the line that prolongs it) describes an arc of an ellipse. This is known as the trammel construction of an ellipse (Eves 1965, p. 177).

Let an ellipse lie along the x-axis and find the equation of the figure (1) where F_1 and F_2 are at $(-c, 0)$ and $(c, 0)$. In Cartesian coordinates,

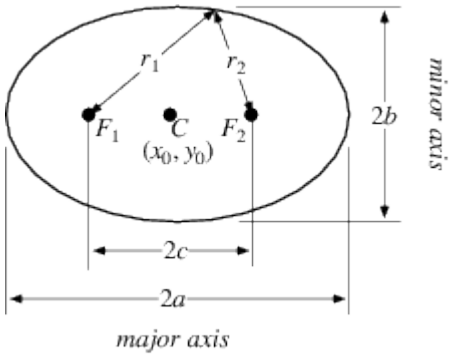
$$\sqrt{(x+c)^2 + y^2} + \sqrt{(x-c)^2 + y^2} = 2a.$$

Bring the second term to the right side and square both sides,

$$(x+c)^2 + y^2 = 4a^2 - 4a\sqrt{(x-c)^2 + y^2} + (x-c)^2 + y^2.$$

Now solve for the square root term and simplify

$$\begin{aligned}\sqrt{(x-c)^2 + y^2} &= -\frac{1}{4a}\left(x^2 + 2xc + c^2 + y^2 - 4a^2 - x^2 + 2xc - c^2 - y^2\right) \\ &= -\frac{1}{4a}\left(4xc - 4a^2\right)\end{aligned}$$



$$= a - \frac{c}{a} x.$$

Square one final time to clear the remaining square root,

$$x^2 - 2xc + c^2 + y^2 = a^2 - 2cx + \frac{c^2}{a^2} x^2.$$

Grouping the x terms then gives

$$x^2 \frac{a^2 - c^2}{a^2} + y^2 = a^2 - c^2,$$

which can be written in the simple form

$$\frac{x^2}{a^2} + \frac{y^2}{a^2 - c^2} = 1.$$

Defining a new constant

$$b^2 \equiv a^2 - c^2$$

puts the equation in the particularly simple form

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1.$$

Conclusion :

MATLAB Program

```
a=input('Enter Horizontal radius :  '); % horizontal radius
b=input('Enter Vertical radius :  '); % vertical radius
Z=input('Enter the centere coordinates as [x,y] : ');
x0=Z(1); % x0,y0 ellipse centre coordinates
y0=Z(2);
t=-pi:0.01:pi;
x=x0+a*cos(t);
y=y0+b*sin(t);
plot(x,y,'r')
hold on
plot(x0,y0,'or');
grid on
title('Scan Convergence of Ellip')
xL = xlim;
yL = ylim;
line([0 0], yL);  %y-axis
line(xL, [0 0]);  %x-axis
hold off
```

Experiment No 6

Scan convergence of Rectangle & Square

Objective : Drawing rectangle/square using MATLAB script.

Theory : In Euclidean plane geometry, a rectangle is a quadrilateral with four right angles. It can also be defined as an equiangular quadrilateral, since equiangular means that all of its angles are equal (360°/4 = 90°). It can also be defined as a parallelogram containing a right angle. A rectangle with four sides of equal length is a square. The term oblong is occasionally used to refer to a non-square rectangle. A rectangle with vertices ABCD would be denoted as Rectanglen.PNG ABCD.

The word rectangle comes from the Latin rectangulus, which is a combination of rectus (as an adjective, right, proper) and angulus (angle).

A crossed rectangle is a crossed (self-intersecting) quadrilateral which consists of two opposite sides of a rectangle along with the two diagonals.[4] It is a special case of an antiparallelogram, and its angles are not right angles. Other geometries, such as spherical, elliptic, and hyperbolic, have so-called rectangles with opposite sides equal in length and equal angles that are not right angles.

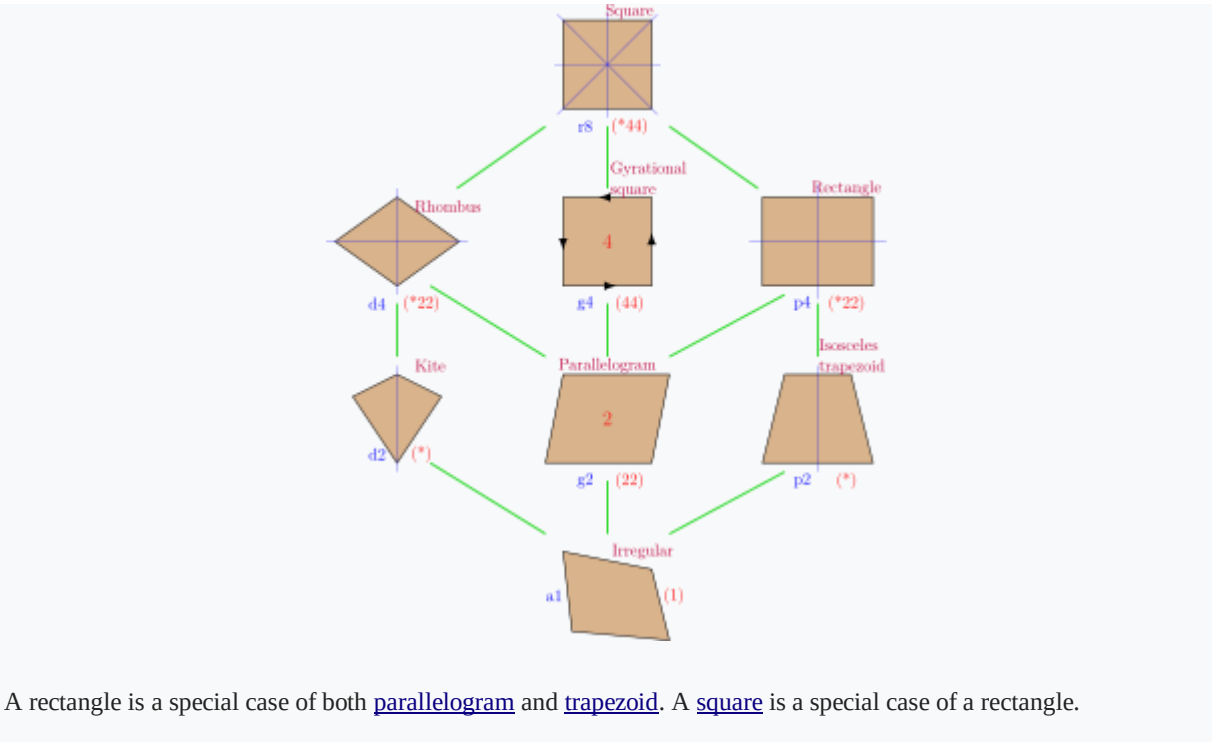
Rectangles are involved in many tiling problems, such as tiling the plane by rectangles or tiling a rectangle by polygons.

Characterizations

A convex quadrilateral is a rectangle if and only if it is any one of the following:[5][6]

- a parallelogram with at least one right angle
- a parallelogram with diagonals of equal length
- a parallelogram *ABCD* where triangles *ABD* and *DCA* are congruent
- an equiangular quadrilateral
- a quadrilateral with four right angles
- a convex quadrilateral with successive sides *a, b, c, d* whose area is 1/4(a+c)(b+d)
- a convex quadrilateral with successive sides *a, b, c, d* whose area is $1/2\sqrt{(a^2+c^2)(b^2+d^2)}$

Classification



A rectangle is a special case of both parallelogram and trapezoid. A square is a special case of a rectangle.

Traditional hierarchy

A rectangle is a special case of a parallelogram in which each pair of adjacent sides is perpendicular.

A parallelogram is a special case of a trapezium (known as a trapezoid in North America) in which *both* pairs of opposite sides are parallel and equal in length.

A trapezium is a convex quadrilateral which has at least one pair of parallel opposite sides.

A convex quadrilateral is

- **Simple**: The boundary does not cross itself.
- **Star-shaped**: The whole interior is visible from a single point, without crossing any edge.

Alternative hierarchy

De Villiers defines a rectangle more generally as any quadrilateral with axes of symmetry through each pair of opposite sides.[8]This definition includes both right-angled rectangles and crossed rectangles. Each has an axis of symmetry parallel to and equidistant from a pair of opposite sides, and another which is the perpendicular bisector of those sides, but, in the case of the crossed rectangle, the first axis is not an axis of symmetry for either side that it bisects.

Quadrilaterals with two axes of symmetry, each through a pair of opposite sides, belong to the larger class of quadrilaterals with at least one axis of symmetry through a pair of opposite sides. These quadrilaterals comprise isosceles trapezia and crossed isosceles trapezia (crossed quadrilaterals with the same vertex arrangement as isosceles trapezia).

Conclusion :

Experiment No 7

McLaurin Series

Objective : Drawing ellipse using MATLAB script

Theory :

1. Basics and examples

Consider a function f defined by a power series of the form

$$f(x) = \sum_{n=0}^{\infty} c_n (x-a)^n, \tag{1}$$

with radius of convergence $R > 0$. If we write out the expansion of $f(x)$ as

$$f(x) = c_0 + c_1(x-a) + c_2(x-a)^2 + c_3(x-a)^2 + c_4(x-a)^4 \ldots,$$

we observe that $f(a) = c_0$. Moreover

$$\begin{aligned} f'(x) &= c_1 + 2c_2(x-a) + 3c_3(x-a)^2 + 4c_4(x-a)^3 + \ldots, \\ f^{(2)}(x) &= 2c_2 + 2\cdot 3 \cdot c_3(x-a) + 3\cdot 4 \cdot c_4(x-a)^2 + \ldots \\ f^{(3)}(x) &= 2\cdot 3 \cdot c_3 + 2\cdot 3 \cdot 4 \cdot c_4(x-a)^2 + \ldots \end{aligned}$$

After computing the above derivatives we observe that

$$\begin{aligned} f(a) &= c_0, \\ f'(a) &= c_1, \\ f^{(2)}(a) = 2 &\Longrightarrow c_2 = \frac{f^{(2)}(a)}{2!}, \\ f^{(3)}(a) = 2\cdot 3 \cdot c_3 &\Longrightarrow c_3 = \frac{f^{(3)}(a)}{3!}. \end{aligned}$$

In general we have

$$f^{(n)}(a) = n! c_n \Longrightarrow c_n = \frac{f^{(n)}(a)}{n!},$$

We have shown the following

Theorem 1 (Taylor-Maclaurin series). *Suppose that $f(x)$ has a power series expansion at $x = a$ with radius of convergence $R > 0$, then the series expansion of $f(x)$ takes the form*

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n = f(a) + f'(a)(x-a) + \frac{f^{(2)}(a)}{2!}(x-a)^2 + \ldots, \tag{2}$$

that is, the coefficient c_n *in the expansion of* $f(x)$ *centered at* $x = a$ *is precisely* $c_n = \dfrac{f^{(n)}(a)}{n!}$. *The expansion* (2) *is called* **Taylor series**. *If* $a = 0$, *the expansion*

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(0)}{n!} x^n = f(0) + f'(0)x + \frac{f^{(2)}(0)}{2!} x^2 + \ldots,$$

is called **Maclaurin Series**.

Let us now consider several classical Taylor series expansions. For the following examples we will assume that all of the functions involved can be expanded into power series.

Conclusion :

Experiment No 8

Bisection method

Objective : Write a program for bisection method in MATLAB

Theory :

There are various methods available for finding the roots of given equation such as Bisection method, False position method, Newton-Raphson method, etc. Today I am going to explain Bisection method for finding the roots of given equation. I will also explain MATLAB program for Bisection method. Bisection method is very simple but time-consuming method. In this method, we first define an interval in which our solution of the equation lies. As the name indicates, Bisection method uses the bisecting (divide the range by 2) principle. In this method, we minimize the range of solution by dividing it by integer 2.

Following are the steps to find the approximate solution of given equation using Bisection method:

Let us assume that we have to find out the roots of f(x), whose solution lies in the range (a,b), which we have to determine. The only condition for bisection method is that f(a) and f(b) should have opposite signs (f(a) negative and f(b) positive). When f(a) and f(b) are of opposite signs at least one real root between 'a' and 'b' should exist.

For the first approximation, we assume that root to be,

x0=(a+b)/2

Then we have to find a sign of f(x0).
If f(x0) is negative the root lies between a and x0. If f(x0) is positive the root lies between x0 and b.
Now we have new minimized range, in which our root lies.
The next approximation is given by,

- x1 = (a+x0)/2…….if f(x0) is negative
- x1 = (x0+b)/2…….if f(x0) is positive

In this taking midpoint of the range of approximate roots, finally, both values of range converge to a single value, which we can take as an approximate root.

Algorithm for Bisection Method:

1. Input function and limits.
2. Repeat steps 3 and 4 100 times.
3. x=(a+b)/2
4. If f(x0)<0, a=x else b=x
5. Display x
6. Repeat steps 7 and 8 10 times.
7. Error = x-(a+b)/2
8. Store error values in array
9. Plot error
10. STOP.

MATLAB PROGRAM

```
function p = bisection(f,a,b)
% provide the equation you want to solve with R.H.S = 0
form.
% Write the L.H.S by using inline function
% Give initial guesses.
% Solves it by method of bisection.
% A very simple code. But may come handy
if f(a)*f(b)>0
    disp('Wrong choice bro')
else
    p = (a + b)/2;
    err = abs(f(p));
    while err > 1e-7
   if f(a)*f(p)<0
       b = p;
   else
       a = p;
   end
    p = (a + b)/2;
   err = abs(f(p));
    end
end
```

Experiment No 9

Simpsons 1/3 Rule for Inegration

Objective : WRITING MATLAB CODE FOR SIMPSONS 1/3 RULE FOR INTEGRATION

Theory :

This function computes the integral "I" via Simpson's rule in the interval [a,b] with n+1 equally spaced points

Syntax: I = simpsons(f,a,b,n)

Where,
f= can either be an anonymous function (e.g. f=@(x) sin(x)) or a vector containing equally spaced values of the function to be integrated
a= Initial point of interval
b= Last point of interval
n= # of sub-intervals (panels), must be integer

Written by Juan Camilo Medina - The University of Notre Dame
09/2010 (copyright Dr. Simpson)

Example 1:

Suppose you want to integrate a function f(x) in the interval [-1,1].
You also want 3 integration points (2 panels) evenly distributed through the domain (you can select more point for better accuracy).
Thus:

```
f=@(x) ((x-1).*x./2).*((x-1).*x./2);
I=simpsons(f,-1,1,2)
```

Example 2:

Suppose you want to integrate a function f(x) in the interval [-1,1].
You know some values of the function f(x) between the given interval, those are fi= {1,0.518,0.230,0.078,0.014,0,0.006,0.014,0.014,0.006,0}
Thus:

```
fi= [1 0.518 0.230 0.078 0.014 0 0.006 0.014 0.014 0.006 0];
I=simpsons(fi,-1,1,[])
```

note that there is no need to provide the number of intervals (panels) "n", since they are implicitly specified by the number of elements in the vector fi

Program in MATLAB

```
function I = simpsons(f,a,b,n)
% This function computes the integral "I" via Simpson's rule in the
interval [a,b] with n+1 equally spaced points
%
% Syntax: I = simpsons(f,a,b,n)
%
% Where,
%  f= can be either an anonymous function (e.g. f=@(x) sin(x)) or a vector
%  containing equally spaced values of the function to be integrated
%  a= Initial point of interval
%  b= Last point of interval
%  n= # of sub-intervals (panels), must be integer
%
%  Written by Juan Camilo Medina  - The University of Notre Dame
%  09/2010 (copyright Dr. Simpson)
%
%
% Example 1:
%
% Suppose you want to integrate a function f(x) in the interval [-1,1].
% You also want 3 integration points (2 panels) evenly distributed through
the
% domain (you can select more point for better accuracy).
% Thus:
%
% f=@(x) ((x-1).*x./2).*((x-1).*x./2);
% I=simpsons(f,-1,1,2)
%
%
% Example 2:
%
% Suppose you want to integrate a function f(x) in the interval [-1,1].
% You know some values of the function f(x) between the given interval,
% those are fi= {1,0.518,0.230,0.078,0.014,0,0.006,0.014,0.014,0.006,0}
% Thus:
%
% fi= [1 0.518 0.230 0.078 0.014 0 0.006 0.014 0.014 0.006 0];
% I=simpsons(fi,-1,1,[])
%
% note that there is no need to provide the number of intervals (panels)
"n",
% since they are implicitly specified by the number of elements in the
% vector fi
if numel(f)>1 % If the input provided is a vector
    n=numel(f)-1; h=(b-a)/n;
    I= h/3*(f(1)+2*sum(f(3:2:end-2))+4*sum(f(2:2:end))+f(end));
else % If the input provided is an anonymous function
    h=(b-a)/n; xi=a:h:b;
    I= h/3*(f(xi(1))+2*sum(f(xi(3:2:end-
2)))+4*sum(f(xi(2:2:end)))+f(xi(end)));
end
```

Experiment No 10

Newton Rapson Method

Objective : Writing Code for Newton Rapson method

Theory :

This article is about Newton's method for finding roots. For Newton's method for finding minima, see Newton's method in optimization.

In numerical analysis, Newton's method (also known as the Newton–Raphson method), named after Isaac Newton and Joseph Raphson, is a method for finding successively better approximations to the roots (or zeroes) of a real-valued function. It is one example of a root-finding algorithm.

The Newton–Raphson method in one variable is implemented as follows:

The method starts with a function f defined over the real numbers x, the function's derivative f ′, and an initial guess x0 for a root of the function f. If the function satisfies the assumptions made in the derivation of the formula and the initial guess is close, then a better approximation x1 is

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Geometrically, (x1, 0) is the intersection of the x-axis and the tangent of the graph of f at (x0, f (x0)).

The process is repeated as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

until a sufficiently accurate value is reached.

This algorithm is first in the class of Householder's methods, succeeded by Halley's method. The method can also be extended to complex functions and to systems of equations.

Program :

```
%The Newton Raphson Method
clc;
close all;
clear all;
syms x;
f=x*exp(x)-1; %Enter the Function here
g=diff(f); %The Derivative of the Function
n=input('Enter the number of decimal places:');
epsilon = 5*10^-(n+1)
x0 = input('Enter the intial approximation:');
for i=1:100
     f0=vpa(subs(f,x,x0)); %Calculating the value of
function at x0
     f0_der=vpa(subs(g,x,x0)); %Calculating the value of
function derivative at x0
  y=x0-f0/f0_der; % The Formula
err=abs(y-x0);
if err<epsilon %checking the amount of error at each
iteration
break
end
x0=y;
end
y = y - rem(y,10^-n); %Displaying upto required decimal
places
fprintf('The Root is : %f \n',y);
fprintf('No. of Iterations : %d\n',i);
```