MAT2001 – Numerical Methods for Engineers

MATLAB Report

Prepared by

Name: Amit Kumar Sahu

Reg No: 18MIS7250

Submitted to

Dr Satyanarayana Badeti

Course Instructor

MATLAB Experiment No – 1

Name: Amit Kumar Sahu **Reg No:** 18MIS7250

Aim: To implement Newton's Raphson method for a system for linear system of equation.

Objective: we have to solve a nonlinear equation, $f(x)=0$ that we cannot easily solve analytically.

Algorithm :

This is a very popular method that usually converges rapidly. It solves the equation $f(x)=0$, assuming that we can compute $f'(x)$. The iterations start with an initial guess x_0 and proceeds as

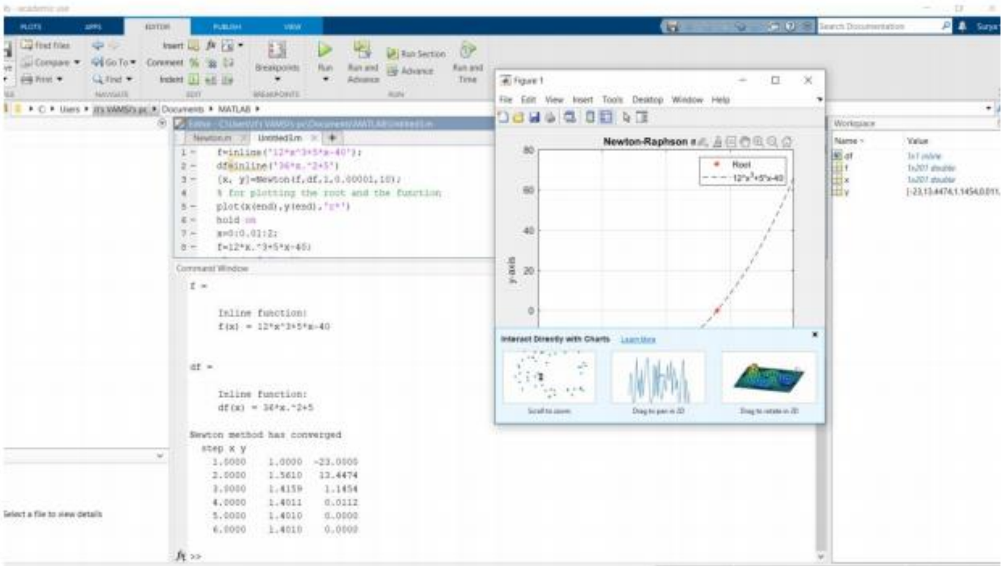
$$x_{k+1}=x_k-\{f(x_k)/f'(x_k)\}.$$

MATLAB Code:

```
function [x,y]=Newton(fun,funpr,x1,tol,kmax)

x(1)=x1;

y(1)=feval(fun,x(1));

ypr(1)=feval(funpr,x(1));

for k=2:kmax

 x(k)=x(k-1)-y(k-1)/ypr(k-1);

 y(k)=feval(fun,x(k));

 if abs(x(k)-x(k-1))<tol

 disp('Newton method has converged');

 break;

 end

 ypr(k)=feval(funpr,x(k));

 iter=k;

end

if(iter>=kmax)

 disp('zero not found to desired tolerance');

end
```

```
n=length(x);

k=1:n;

out=[k' x' y'];

disp(' step x y')

disp(out)

Untitled3.m

f=inline('12*x^3+5*x-40')

df=inline('36*x.^2+5')

[x, y]=Newton(f,df,1,0.00001,10);

% for plotting the root and the functionplot(x(end),y(end),'r*')

hold on

x=0:0.01:2;

f=12*x.^3+5*x-40;

plot(x,f,'k--')

grid on

xlabel('x-axis')

ylabel('y-axis')

title('Newton-Raphson method')

legend('Root','12*x^3+5*x-40')
```

Output:



Graph:

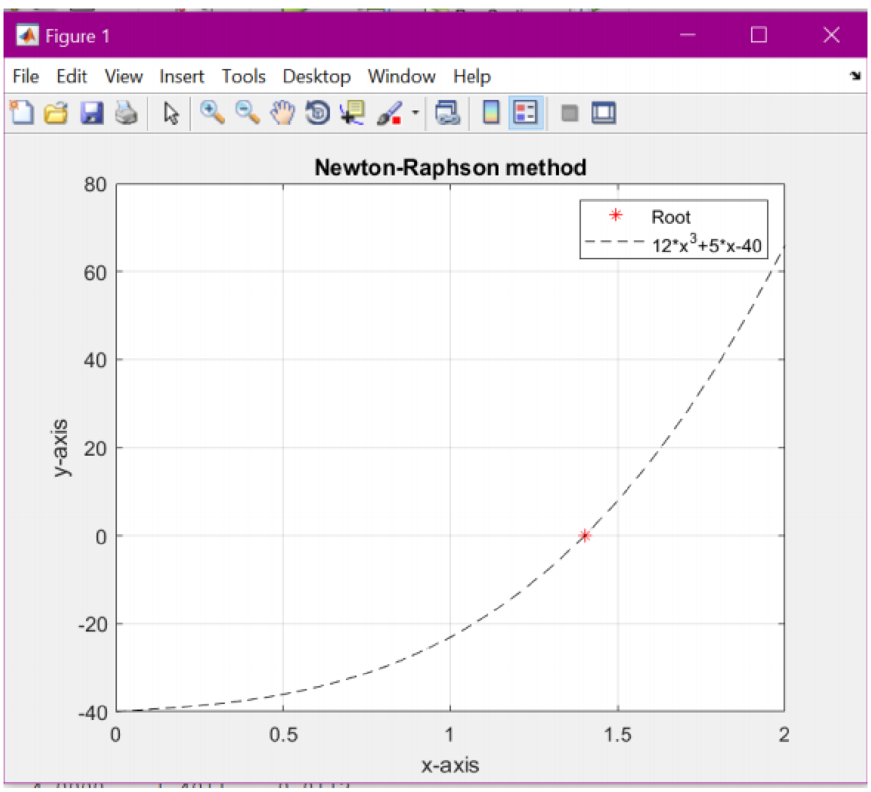


Figure 1
File Edit View Insert Tools Desktop Window Help
Newton-Raphson method
Root
12*x^3+5*x-40
y-axis
x-axis
80
60
40
20
0
-20
-40
0
0.5
1
1.5
2

MATLAB Experiment No – 2

Name: Amit Kumar Sahu **Reg No:** 18MIS7250

Aim: To implement Newton's method for a system of three equations

Objective: we have to solve a nonlinear equation, $f(x)=0$ that we cannot easily solve analytically.

Algorithm :

This is a very popular method that usually converges rapidly. It solves the equation $f(x)=0$, assuming that we can compute $f'(x)$. The iterations start with an initial guess x_0 and proceeds as

$$x_{k+1}=x_k-\{f(x_k)/f'(x_k)\}.$$

MATLAB Code:

```
function x = NewtonSys(F,J,x0,tol,kmax)
xold=x0; iter=1;
while(iter<=kmax)
    y=-feval(J,xold)\feval(F,xold);
    xnew=xold+y';
    dif=norm(xnew-xold);
    disp([iter xnew dif]);
    if dif<=tol
        x=xnew;
        disp('Newton method has converged')
 return;
    else
        xold=xnew;
    end
    iter=iter+1
end
disp('Newton method has converged')
x=xnew


disp("For A=1 & B=1 the values are")
F=inline('[1+x(1)^2*x(2)-2*x(1); x(1)-x(1)^2*x(2)]');
F1=inline('[1+x(1)^2*x(2)-4*x(1); 3*x(1)-x(1)^2*x(2)]');
F2=inline('[1+x(1)^2*x(2)-3*x(1); 2*x(1)-x(1)^2*x(2)]');
J=inline('[2*x(1)*x(2) - 2,  x(1)^2;1 - 2*x(1)*x(2), -x(1)^2]')
x0=[1 1]; tol=0.0001;kmax=20;
disp("For A=1 & B=1 the values are")
x=NewtonSys(F,J,x0,tol,kmax)
disp("For A=1 & B=3 the values are")
x1=NewtonSys(F1,J,x0,tol,kmax)
disp("For A=1 & B=2 the values are")
x2=NewtonSys(F2,J,x0,tol,kmax)
```

Output:

```
>> runningnewtonsysmethod
For A=1 & B=1 the values are

J =

     Inline function:
     J(x) = [2*x(1)*x(2) - 2,  x(1)^2;1 - 2*x(1)*x(2), -x(1)^2]

For A=1 & B=1 the values are
     1     1     1     0

Newton method has converged

x =

     1     1

For A=1 & B=3 the values are
     1     1     3     2


iter =

     2

     2     1     3     0

Newton method has converged

x1 =

     1     3

For A=1 & B=2 the values are
     1     1     2     1


iter =

     2

     2     1     2     0

Newton method has converged

x2 =

     1     2
```

MATLAB Experiment No – 3

Name: Amit Kumar Sahu **Reg No:** 18MIS7250

Aim: To solve the system secant method

Objective: To find root r that uses a succession of roots of secant lines to better approximate a root of a function *f*.

Algorithm : The secant method is defined by the recurrence relation

$$x_n = x_{n-1} - f(x_{n-1})\frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})} = \frac{x_{n-2}f(x_{n-1}) - x_{n-1}f(x_{n-2})}{f(x_{n-1}) - f(x_{n-2})}.$$

As can be seen from the recurrence relation, the secant method requires two initial values, x_0 and x_1, which should ideally be chosen to lie close to the root.

MATLAB Code:

```
function [xx, yy]=Secant(f,a,b,tol,kmax)
y(1)=f(a);
y(2)=f(b);
x(1)=a;
x(2)=b;
Dx(1)=0;
Dx(2)=0;
disp(' step x(k-1) x(k) x(k+1) y(k+1) Dx(k+1)')
for k=2:kmax
 x(k+1)=x(k)-y(k)*(x(k)-x(k-1))/(y(k)-y(k-1));
 y(k+1)=f(x(k+1));
 Dx(k+1)=x(k+1)-x(k);
 iter=k-1;
 out=[iter, x(k-1),x(k),x(k+1),y(k+1),Dx(k+1)];
 disp(out)
 xx=x(k+1);
 yy=y(k+1);
 if abs(y(k+1))<tol
 disp('Secant method has converged'); break;
 end
 if (iter>=kmax)
 disp('zero not found to desired tolerance');
 end
end


f=@(x) 2*x.^2+3*log(x)-1;
```

```
a=1;b=2;
tol=0.00001;kmax=10;
[xx, yy]=Secant(f,a,b,tol,kmax);
x=0:0.01:3;
y=2*x.^2+3*log(x)-1;
plot(x,y)
hold on
plot(xx(end),yy(end),'r*')
hold on
xlabel('X-Axis')
ylabel('Y-Axis')
title('Secant Method')
```

Output:

step x(k-1) x(k) x(k+1) y(k+1) Dx(k+1

1.0000 0.5000 1.0000 0.8603 0.0289 -0.1397

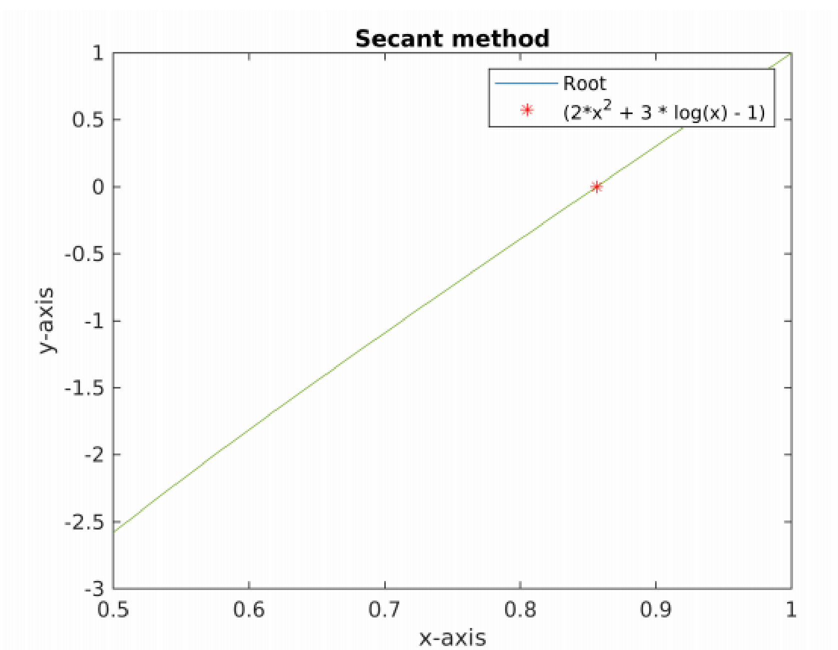
2.0000 1.0000 0.8603 0.8562 0.0001 -0.0042

3.0000 0.8603 0.8562 0.8561 -0.0000 -0.0000

secant method has converged

0.8561 -0.0000

Graph:



MATLAB Experiment No – 4

Name: Amit Kumar Sahu **Reg No:** 18MIS7250

Aim: To solve the system using Gauss Elimination

Objective: Solving systems of linear equations. It consists of a sequence of operations performed on the corresponding matrix of coefficients

Algorithm: Gaussian elimination allows the computation of the determinant of a square matrix, we have to recall how the elementary row operations change the determinant:

- Swapping two rows multiplies the determinant by −1
- Multiplying a row by a nonzero scalar multiplies the determinant by the same scalar
- Adding to one row a scalar multiple of another does not change the determinant.

If Gaussian elimination applied to a square matrix *A* produces a row echelon matrix *B*, let *d* be the product of the scalars by which the determinant has been multiplied, using the above rules. Then the determinant of *A* is the quotient by *d* of the product of the elements of the diagonal of *B*:

$$\det(A) = \frac{\prod \operatorname{diag}(B)}{d}.$$

MATLAB Code:

```
function x=Gaussele(A,b);
A=[1 3 5;2 -1 -3;4 5 -1];
b=[14;3;7];
[m,n]=size(A);
if m~=n
    error('Matrix A must be square');
end
nb=n+1;
Aug=[A b];

for k=1:n-1
    for i=k+1:n
        factor=Aug(i,k)/Aug(k,k);
        Aug(i,k:nb)=Aug(i,k:nb)-factor*Aug(k,k:nb);
    end
end
```

```
x=zeros(n,1);
x(n)=Aug(n,nb)/Aug(n,n);
for i=n-1:-1:1
    x(i)=(Aug(i,nb)-Aug(i,i+1:n)*x(i+1:n))/Aug(i,i);
end
disp('Gauss elimination method')
x(i);
```

Output:

```
gaussian_method
Gauss elimination method

ans =

     5
    -2
     3
```

MATLAB Experiment No – 5

Name: Amit Kumar Sahu **Reg No:** 18MIS7250

Aim: To solve the system Gauss Siedel method

Objective: Iteratively solving the system of linear equations

Algorithm:

```
Inputs: A, b

Output:


Choose an initial guess       to the solution
repeat until convergence
    for i from 1 until n do


        for j from 1 until n do
            if j ≠ i then


            end if
        end (j-loop)


    end (i-loop)
    check if convergence is reached
end (repeat)
```

MATLAB Code:

```
clear ; clc ; close all
n = input('size of the equation system  n =   ') ;
C = input('Matrix C ' ) ;
b = input('Matrix b ' ) ;
dett = det(C)
if dett == 0
    print('cannot solve because det(C) = 0 ')
else
b = b'
A = [ C  b ]
for j = 1:(n-1)
        for i= (j+1) : n
            mult = A(i,j)/A(j,j) ;
            for k= j:n+1
                A(i,k) = A(i,k) - mult*A(j,k) ;
                A
```

```
            end
        end
end
for p = n:-1:1
    for r = p+1:n
        x(p) = A(p,r)/A(p,r-1)
    end
   end
end
```

Output:

size of the equation system n =

3

Matrix C

[6 -2 1;1 2 -5;-2 7 2]

Matrix b

[0 0 0]

dett =

  229.0000

b =

```
     0
     0
     0
```

A =

```
     6    -2     1     0
     1     2    -5     0
    -2     7     2     0
```

A =

```
     6    -2     1     0
     0     2    -5     0
    -2     7     2     0


A =
    6.0000   -2.0000    1.0000         0
         0    2.3333   -5.0000         0
   -2.0000    7.0000    2.0000         0


A =
    6.0000   -2.0000    1.0000         0
         0    2.3333   -5.1667         0
   -2.0000    7.0000    2.0000         0


A =
    6.0000   -2.0000    1.0000         0
         0    2.3333   -5.1667         0
   -2.0000    7.0000    2.0000         0


A =
    6.0000   -2.0000    1.0000         0
         0    2.3333   -5.1667         0
         0    7.0000    2.0000         0


A =
    6.0000   -2.0000    1.0000         0
         0    2.3333   -5.1667         0
         0    6.3333    2.0000         0


A =
    6.0000   -2.0000    1.0000         0
         0    2.3333   -5.1667         0
         0    6.3333    2.3333         0


A =
```

```
    6.0000   -2.0000    1.0000         0
         0    2.3333   -5.1667         0
         0    6.3333    2.3333         0


A =

    6.0000   -2.0000    1.0000         0
         0    2.3333   -5.1667         0
         0         0    2.3333         0


A =

    6.0000   -2.0000    1.0000         0
         0    2.3333   -5.1667         0
         0         0   16.3571         0


A =

    6.0000   -2.0000    1.0000         0
         0    2.3333   -5.1667         0
         0         0   16.3571         0


x =

         0   -2.2143


x =

   -0.3333   -2.2143


x =

   -0.5000   -2.2143
```

MATLAB Experiment No – 6

Name: Amit Kumar Sahu **Reg No:** 18MIS7250

Aim: To solve the system Jacobi method

Objective: For determining the solutions of a strictly diagonally dominant system of linear equations.

Algorithm :

Input: initial guess $x^{(0)}$ to the solution, (diagonal dominant) matrix A, right-hand side vector b, convergence criterion
Output: solution when convergence is reached
Comments: pseudocode based on the element-based formula above

$k = 0$
while convergence not reached **do**
  for i := 1 **step until** n **do**
    $\sigma = 0$
    for j := 1 **step until** n **do**
      if j ≠ i **then**
        $\sigma = \sigma + a_{ij} x_j^{(k)}$
      end
    end
    $x_i^{(k+1)} = \frac{1}{a_{ii}} (b_i - \sigma)$
  end
  $k = k + 1$
end

MATLAB Code:

```
clc;
clear all;
A=[5 -2 3;-3 9 1;2 -1 -7]
b=[-1;2;3]
N=40
x=[1,1,1]
jacobi(A, b, N)

function jacobi(A, b, N)
test=all((2*abs(diag(A)))- sum(abs(A),2)>=0);
if test==0
    A([1 2],:) = A([2 1],:);
    b([1 2]) = b([2 1]);
end

test=all((2*abs(diag(A)))- sum(abs(A),2)>=0);

if test==0
    A([2 1],:) = A([1 2],:);
    b([2 1]) = b([1 2]);
    A([1 3],:) = A([3 1],:);
    b([1 3]) = b([3 1]);
    disp("not a dominant vector")
```

```
end
disp(" dominant vector")

d=diag(A);
D=diag(d);
disp("Displaying the diagonal matrix")
disp(D)
D_inv=inv(D);
disp("Displaying the inverse of diagonal matrix")
disp(D_inv)
E=A-D;
disp("Displaying remainder matrix")
disp(E)
x=[1;1;1];
T=-D_inv*E;
C=D_inv*b;

for j=1:N
    x=T*x+C;
end
disp("Here are the result of the following matrix: ")
disp(x)
end
```

Output:

```
A =

     5    -2     3
    -3     9     1
     2    -1    -7


b =

    -1
     2
     3


N =

    40


x =
```

```
     1     1     1

 dominant vector
Displaying the diagonal matrix
     5     0     0
     0     9     0
     0     0    -7

Displaying the inverse of diagonal matrix
    0.2000         0         0
         0    0.1111         0
         0         0   -0.1429

Displaying remainder matrix
     0    -2     3
    -3     0     1
     2    -1     0

Here are the result of the following matrix:
    0.1861
    0.3312
   -0.4227
```

MATLAB Experiment No – 7

Name: Amit Kumar Sahu **Reg No:** 18MIS7250

Aim: To solve the system using LU Decomposition

Objective: Factors a matrix as the product of a lower triangular matrix and an upper triangular matrix.

Algorithm:

Let *A* be a square matrix. An **LU factorization** refers to the factorization of *A*, with proper row and/or column orderings or permutations, into two factors – a lower triangular matrix *L* and an upper triangular matrix *U*:

A=LU

In the lower triangular matrix all elements above the diagonal are zero, in the upper triangular matrix, all the elements below the diagonal are zero. For example, for a 3 × 3 matrix *A*, its LU decomposition looks like this:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

Without a proper ordering or permutations in the matrix, the factorization may fail to materialize.

For example, it is easy to verify (by expanding the matrix multiplication) that . If , then

at least one of and has to be zero, which implies that either *L* or *U* is singular. This is impossible if *A* is nonsingular (invertible). This is a procedural problem. It can be removed by simply reordering the rows of *A* so that the first element of the permuted matrix is nonzero. The same problem in subsequent factorization steps can be removed the same way; see the basic procedure below.

MATLAB Code:

```
clc;
clear all;
A = [10 -7 0
     -3  2 6
      5 -1 5];
[L,U] = lu(A)
disp("calculating L*U")
```

```
L*U
[L,U,P] = lu(A)
disp("calculating P'*L*U")
P'*L*U
```

Output:

```
L =

    1.0000         0         0
   -0.3000   -0.0400    1.0000
    0.5000    1.0000         0


U =

   10.0000   -7.0000         0
         0    2.5000    5.0000
         0         0    6.2000

calculating L*U

ans =

   10.0000   -7.0000         0
   -3.0000    2.0000    6.0000
    5.0000   -1.0000    5.0000


L =

    1.0000         0         0
    0.5000    1.0000         0
   -0.3000   -0.0400    1.0000


U =

   10.0000   -7.0000         0
```

```
         0    2.5000    5.0000
         0         0    6.2000


P =

     1     0     0
     0     0     1
     0     1     0

calculating P'*L*U

ans =

   10.0000   -7.0000         0
   -3.0000    2.0000    6.0000
    5.0000   -1.0000    5.0000
```

MATLAB Experiment No – 8

Name: Amit Kumar Sahu **Reg No:** 18MIS7250

Aim: Implementing Power Method.

Objective: The algorithm will produce a number Lambda, which is the greatest (in absolute value) eigenvalue of A, and a nonzero vector v, which is a corresponding eigenvector of Lambda, that is, Av=(Lambda)*(v).

Algorithm: The power iteration algorithm starts with a vector b0, which may be an approximation to the dominant eigenvector or a random vector. The method is described by the recurrence relation

$$b_{k+1} = \frac{Ab_k}{\|Ab_k\|}$$

MATLAB Code:

```
n=input('Enter dimension of the matrix, n:  ');
A = zeros(n,n);
x = zeros(1,n);
y = zeros(1,n);
tol = input('Enter the tolerance, tol: ');
m = input('Enter maximum number of iterations, m:  ');


A=[1 2 0; -2 1 2; 1 3 1];
x=[1 1 1];


k = 1; lp = 1;
amax = abs(x(1));
for i = 2 : n
   if abs(x(i)) > amax
       amax = abs(x(i));
       lp = i;
   end
end
for i = 1 : n
   x(i) = x(i)/amax;
 end

fprintf('\n\n  Ite.    Eigenvalue     ............Eigenvectores............\n');
while k <= m
     for i = 1 : n
        y(i) = 0;
```

```
        for j = 1 : n
          y(i) = y(i) + A(i,j) * x(j);
        end
     end
     ymu = y(lp);
     lp = 1;
     amax = abs(y(1));
     for i = 2 : n
        if abs(y(i)) > amax
           amax = abs(y(i));
           lp = i;
        end
     end
     if amax <= 0
        fprintf('0 eigenvalue - select another ');
        fprintf('initial vector and begin again\n');
     else
          err = 0;
          for i = 1 : n
             t = y(i)/y(lp);
             if abs(x(i)-t) > err
               err = abs(x(i)-t);
             end
             x(i) = t;
          end
          fprintf('%4d     %11.8f', k, ymu);
          for i = 1 : n
            fprintf('   %11.8f', x(i));
          end
          fprintf('\n');
          if err <= tol
             fprintf('\n\nThe eigenvalue after %d iterations is: %11.8f \n',k, ymu);
             fprintf('The corresponding eigenvector is: \n');
             for i = 1 : n
                fprintf('                                %11.8f \n', x(i));
             end
             fprintf('\n');
             break;
          end
          k = k+1;
      end
end
if k > m
fprintf('Method did not converge within %d iterations\n', m);
end
```

Output:

```
powermethod
Enter dimension of the matrix, n:
3
Enter the tolerance, tol:
```

```
0.001
Enter maximum number of iterations, m:
8


  Ite.    Eigenvalue     ...........Eigenvectores...........
   1      3.00000000    0.60000000    0.20000000    1.00000000
   2      2.20000000    0.45454545    0.45454545    1.00000000
   3      2.81818182    0.48387097    0.54838710    1.00000000
   4      3.12903226    0.50515464    0.50515464    1.00000000
   5      3.02061856    0.50170648    0.49488055    1.00000000
   6      2.98634812    0.49942857    0.49942857    1.00000000
   7      2.99771429    0.49980938    0.50057186    1.00000000
   8      3.00152497    0.50006351    0.50006351    1.00000000


The eigenvalue after 8 iterations is:  3.00152497
The corresponding eigenvector is:
                                  0.50006351
                                  0.50006351
                                  1.00000000
```

MATLAB Experiment No – 9

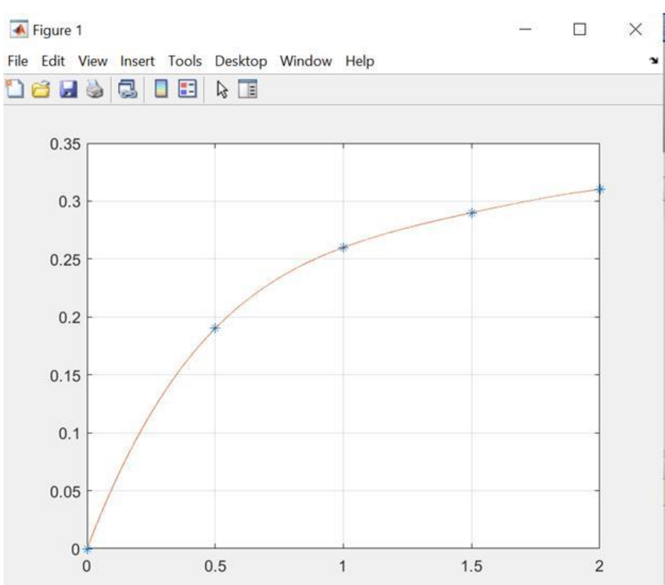
Name: Amit Kumar Sahu **Reg No:** 18MIS7250

Aim: Implementing Lagrange's interpolation.

MATLAB Code:

```
function y=lagrange(x,pointx,pointy)
n=size(pointx,2);
L=ones(n,size(x,2));
if (size(pointx,2)~=size(pointy,2))
    fprintf(1,'\nERROR!\nPOINTX and POINTY must have the same number of elements\n');
    y=NaN;
else
    for i=1:n
        for j=1:n
            if (i~=j)
                L(i,:)=L(i,:).*(x-pointx(j))/(pointx(i)-pointx(j));
            end
        end
    end

y=0;

for i=1:n
    y=y+pointy(i)*L(i,:);
end
end
```

Output:



MATLAB Experiment No – 10

Name: Amit Kumar Sahu **Reg No:** 18MIS7250

Aim: To solve the system using Tridiagonal Method (Thomas Algorithm).

Objective: used to solve tridiagonal systems of equations

Algorithm:

```
Sub TriDiagonal_Matrix_Algorithm(N%, A#(), B#(), C#(), D#(), X#())
    Dim i%, W#
    For i = 2 To N
        W = A(i) / B(i - 1)
        B(i) = B(i) - W * C(i - 1)
        D(i) = D(i) - W * D(i - 1)
    Next i
    X(N) = D(N) / B(N)
    For i = N - 1 To 1 Step -1
        X(i) = (D(i) - C(i) * X(i + 1)) / B(i)
    Next i
End Sub
```

MATLAB Code:

```
%Solving Linear system by using Thomas algorithm /Tridiagonal system
clc;clear all;close all;
format 'short'
%%Triangularization
m=input('Enter the order of TDmatrix:=');%Choose any square matrix
%m=4
% Lower diagonal element such that first entry is zero.
a=input('\n Enter the lower diagonal vector:=')%lower diagonal elements
%a=[0 -1 -1 -1];
b=input('\n Enter the Main diagonal vector:=')%diagonal elements
%b=[2.04 2.04 2.04 2.04];
% Upper diagonal element such that last entry is zero.
c=input('\n Enter the upper diagonal vector:=')%upperdiagonal elements
% c=[-1 -1 -1 0];
d=input('Enter the right side vector:=')
%d=[4.08 0.8 0.8 2.08];
alpha=zeros(1,m);
for i=1:m
    if i==1
        alpha(i)=b(i);
        beta(i)=d(i);
    else
        ivalue=i
        alpha(i)=b(i)-(a(i)/alpha(i-1))*c(i-1);
        beta(i)=d(i)-(a(i)/alpha(i-1))*beta(i-1);
```

```
    end
end
alpha
beta
%% Back substitution
x=zeros(1,m);
for i=m:-1:1
    if i==m
        x(i)=beta(i)/alpha(i);
    else
        x(i)=(beta(i)-c(i)*x(i+1))/alpha(i);
    end
end
x
```

Output:

```
Enter the order of TDmatrix:=
5

 Enter the lower diagonal vector:=
[0 1 1 1 1]

a =

     0     1     1     1     1


 Enter the Main diagonal vector:=
[-2 -2 -2 -2 -2]

b =

    -2    -2    -2    -2    -2


 Enter the upper diagonal vector:=
[1 1 1 1 0]

c =

     1     1     1     1     0
```

```
Enter the right side vector:=
[1;0;0;0;0]

d =

     1
     0
     0
     0
     0


ivalue =

     2


ivalue =

     3


ivalue =

     4


ivalue =

     5


alpha =

   -2.0000   -1.5000   -1.3333   -1.2500   -1.2000


beta =
```

```
    1.0000    0.5000    0.3333    0.2500    0.2000


x =

   -0.8333   -0.6667   -0.5000   -0.3333   -0.1667
```

MATLAB Experiment No – 11

Name: Amit Kumar Sahu **Reg No:** 18MIS7250

Aim: To solve the system of equation using Trapezoidal rule.

Objective: approximating the definite integral.

Algorithm:

$$\int_a^b f(x)\,dx \approx \sum_{k=1}^{N} \frac{f(x_{k-1})+f(x_k)}{2}\Delta x_k = \tfrac{\Delta x}{2}\left(f(x_0)+2f(x_1)+2f(x_2)+2f(x_3)+2f(x_4)+\cdots+2f(x_{N-1})+f(x_N)\right).$$

MATLAB Code:

```
clc;
clear all;
f=@(x)cosh(x);
% create a handle to the function f with an @ sign.
 a=input('Enter lower limit a: ');
 b=input('Enter upper limit b: ');
 n=input('Enter the no. of subinterval: ');
 h=(b-a)/n;
 sum=0;
for k=1:1:n-1
  x(k)=a+k*h;
  y(k)=f(x(k));
  sum=sum+y(k);
end
% Formula:  (h/2)*[(y0+yn)+2*(y2+y3+..+yn-1)]
answer=(h/2)*(f(a)+f(b)+2*sum);
fprintf('\n The value of integration is %f',answer);
```

Output:

```
Enter lower limit a:
0
Enter upper limit b:
2
Enter the no. of subinterval:
4

 The value of integration is 3.702107
```

MATLAB Experiment No – 12

Name: Amit Kumar Sahu **Reg No:** 18MIS7250

Aim: To solve the system using Simpson's 1/3rd Rule

Objective: approximations for definite integrals

Algorithm:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right].$$

MATLAB Code:

```
clc;
clear all;
f=@(x)cosh(x); %Change here for different function
 a=input('Enter lower limit a: ');
 b=input('Enter upper limit b: ');
 n=input('Enter the  number of sub-intervals n: ');
 h=(b-a)/n;
if rem(n,2)==1
   fprintf('\n Enter valid n!!!');
   n=input('\n Enter n as even number ');
end
for k=1:1:n
  x(k)=a+k*h;
  y(k)=f(x(k));
end
 so=0;se=0;
for k=1:1:n-1
    if rem(k,2)==1
       so=so+y(k);%sum of odd terms
     else
       se=se+y(k); %sum of even terms
    end
end
% Formula:  (h/3)*[(y0+yn)+2*(y3+y5+..odd term)+4*(y2+y4+y6+...even terms)]
answer=h/3*(f(a)+f(b)+4*so+2*se);
fprintf('\n The value of integration is %f',answer); % exmple The value of
integration is 0.408009
```

Output:

1

```
Enter upper limit b:
2
Enter the  number of sub-intervals n:
16

 The value of integration is 2.451659
```