Name : Yumna Mubeen

F.Name : Muhammad Nadeem

Section : A

Seat no. : B21110006165

Field : BSCS (4th Sem)

Course : Mathematics IV (Numerical Computing) (406)

Teacher : Ma'am DR Noor Fatima Siddiqui (NFS)

Assignment : 01:

Lab work from 1 to 5 weeks :

Variables:

```
>> a=10

a =

    10

>> b= 12;   % 12 value assigned to b, but by using terminator value of b didn't show after pressing enter.
>> b    % if we write b & enter then value of b will print.

b =

    12

>> clear a  % clear command will remove value of a that we assigned a=10, now it will give error.
>> a
Unrecognized function or variable 'a'.

>> clc % clc command will clear the screen.

>> 10 = a  % (incorrect way)
 10 = a  % (incorrect way)
    ↑
Incorrect use of '=' operator. Assign a value to a variable using '=' and compare values for equality using '=='.
```

Format Command:

```
>> % Format Command:
>> format long    % "long"  will result in 15 decimal places.
>> 2*sin(1.4)

ans =

   1.970899459976920


>> format short % (by default) format is short,will result in 4 decimal places.
>> 2*sin(1.4)

ans =

    1.9709

>> % "format" command can also be used to control the spacing between MATLAB command , expression & result
>> format loose % by default "loose".
>> 5*33

ans =

   165
```

```
>> format compact
>> 5*33
ans =
   165
```

Vector:

```
>> v = [1 2 3 4]      % vector is 1x4 (one row by four columns).
v =
     1     2     3     4
>> nv= 1:2:9     % (first:step:last), integers form 1 to 9 in steps of 2 .
nv =
     1     3     5     7     9

>> new(5)       % index (5) = 9
ans =
     9
>> v =[3 7 2 1];
>> v = v*3
v =
     9    21     6     3
>> g =[3 7 2 1];
>> g = g/2
g =
    1.5000    3.5000    1.0000    0.5000

>> % Vector Addition & Subtraction:
>> v1 = [1 2 3 4];
>> v2 = [5 6 7 8];
>> a= v1+v2

a =

     6     8    10    12

>> b= v1-v2

b =

    -4    -4    -4    -4

>> % Linspace function
>> nv = 1:2:9
nv =
     1     3     5     7     9
>> ls = linspace(3,15,5)   % linspace(x,y,n), create a vector with 'n' values in the inclusive range x to y.
ls =
     3     6     9    12    15
```

Matrix:

```
>> % Creating Column vectors :
>> c = [1;2;3;4]
c =
     1
     2
     3
     4
>> r = c'     % transpose of vector c.
r =
     1     2     3     4
>> mat = [4 3 1 ; 2 5 6]
mat =
     4     3     1
     2     5     6
>> size(mat)
ans =
     2     3
>> % Ones & Zeros in matrix:
>> zeros(3)
ans =
     0     0     0
     0     0     0
     0     0     0
>> ones(2,4)
ans =
     1     1     1     1
     1     1     1     1
>>  % Matrix Addition & Subtraction :
>>  mat1 = [1 2 3; 4 5 6];
>>  mat2 = [7 8 9; 10 11 12];
>> a=mat1-mat2

a =

    -6    -6    -6
    -6    -6    -6

>> b=mat1+mat2

b =

     8    10    12
    14    16    18

>> % Matrix Multiplication :
>> a = [3 8 0; 1 2 5];
>> b = [1 2 3 4; 4 5 1 2; 0 2 3 0];
>> c = a*b     %g = [3 7 2 1];     % for compute multiplication use (*).
c =
    35    46    17    28
     9    22    20     8
```

For Input & Display Message:

```
>> % for input :
>> a = input('Enter your course number :')
Enter your course number :
406
a =
   406
>> input('Hello','s')   % s is the data type (string)
Hello
```

```
>>  % for display any message :
>> disp('MATLAB')
MATLAB
```

Function Handling:

```
>> f = @ (x) x^2 + 1
f =
  function_handle with value:
    @(x)x^2+1
>> f(1)
ans =
     2
```

Programs :

Factorial Program using For Loop :

```
x=input("enter number:");
t =1;
for i=1:x
    t=t*i;
    i =i+1;
end
fprintf("Factorial of %d is = %d",x,t);
```

Output :

```
>> fac_for
enter number:
7
Factorial of 7 is = 5040
```

Factorial Program using While Loop :

```
x=input("enter number:");
i =1;
t =1;
while i <= x
    t=t*i;
    i =i+1;
end
fprintf("Factorial of %d is = %d",x,t);
```

Output :

```
>> fact_while
enter number:
5
Factorial of 5 is = 120
```

Calculate The Median :

```
function [] = Median()
a=input('Enter first number: ');
b=input('Enter second number: ');
c= median([a,b])
fprintf('The median of %d and %d is : %f',a,b,c);
```

Output :

```
>> Median
Enter first number:
5
Enter second number:
7
c =
     6
The median of 5 and 7 is : 6.000000
```

Number Is Even Or Odd :

```
a =input("Enter number: ");
r =rem(a,2);
if (r==0)
    disp("Number is Even. ")
else
    disp("Number is Odd. ")
end
```

Output :

```
>> Even_or_odd
Enter number:
7
Number is Odd.
>> Even_or_odd
Enter number:
6
Number is Even.
```

Sum Of Numbers :

```
function [a] = SumOfNum
a = 0;
for i=1:100
    a=a+i;
end
fprintf("Sum of 100 integers is: %d",a);
end
```

Output :

```
>> SumOfNum
Sum of 100 integers is: 5050
ans =

        5050
```

Sum Of 100 Positive Integers :

```
function [] = SumOfEven
a = 0;
for i=0:2:100  % start from 0 increment by 2 till 100.
    a=a+i;
end
fprintf("Sum of 100 even integers is: %d",a);
end
```

Output :

```
>> SumOfEven
Sum of 100 even integers is: 2550
```

Sum Of 100 Negative Integers :

```
function [] = SumOfOdd
a = 0;
for i=1:2:100  % start from 1 increment by 2 till 100.
    a=a+i;
end
fprintf("Sum of 100 odd integers is: %d",a);
end
```

Output :

```
>> SumOfOdd
Sum of 100 odd integers is: 2500
```

Bisection Method :

```
function[c] = Bisection_Method()
format long
%input the equation as a string.
fu=input('Enter the equation: ','s');
%converting it to an equation.
f=inline(fu);
disp("The equatio is: ")
f
%taking guesses as input
a= input('Enter the first guess: ');
b= input('Enter the second guess: ');
e=input('Enter the tolerance error: ');
n=input('Enter number of iterations: ');
if f(a)*f(b)<0
    for i=1:n
        c=(a+b)/2;
        fprintf('c %d = %.9f\n', i,c)
        if abs(c-b)<e || abs(c-a)<e
            break
        end
        if f(a)*f(c)<0
            b=c;
        elseif f(b)*f(c)<0
            a=c;
        end

    end
else
    disp('No root between the given brackets.' );
end
```

Output :

```
>> Bisection_Method
Enter the equation:
x^3-4*x-9
The equatio is:
f =
     Inline function:
     f(x) = x^3-4*x-9
Enter the first guess:
2
Enter the second guess:
3
Enter the tolerance error:
0.0001
Enter number of iterations:
30
c 1 = 2.500000000
c 2 = 2.750000000
c 3 = 2.625000000
c 4 = 2.687500000
c 5 = 2.718750000
c 6 = 2.703125000
c 7 = 2.710937500
c 8 = 2.707031250
c 9 = 2.705078125
c 10 = 2.706054688
c 11 = 2.706542969
c 12 = 2.706298828
c 13 = 2.706420898
c 14 = 2.706481934
ans =
   2.706481933593750
```

Regula Falsi Method :

```
function [c] = Regula_Falsi_Method()
format long
%input the equation as a string.
fu=input('Enter the equation: ','s');
%converting it to an equation.
f=inline(fu);
disp("The equatio is: ")
f
%taking guesses as input
a= input('Enter the first guess: ');
b= input('Enter the second guess: ');
e=input('Enter the tolerance error: ');
n=input('Enter number of iterations: ');
if f(a)*f(b)<0 && a<b
    for i=1:n
        c=(a*f(b)-b*f(a))/(f(b)-f(a));
        fprintf('c %d = %.4f\n', i,c)
        if abs(f(c))<e
            break
        end
        if f(a)*f(c)<0
            b=c;
        elseif f(b)*f(c)<0
            a=c;
        end

    end
else
    disp('No root between the given brackets.' );
end
```

Output :

```
>> Regula_Falsi_Method
Enter the equation:
x*sin(x)-1
The equatio is:
f =
     Inline function:
     f(x) = x*sin(x)-1
Enter the first guess:
0
Enter the second guess:
2
Enter the tolerance error:
0.0001
Enter number of iterations:
30
c 1 = 1.0998
c 2 = 1.1212
c 3 = 1.1142
ans =
   1.114161194962634
```

Secant Method :

```
function [c] = Secant_Method()
format long
%input the equation as a string.
fu=input('Enter the equation: ','s');
%converting it to an equation.
f=inline(fu);
disp("The equatio is: ")
f
%taking guesses as input
a= input('Enter the first guess: ');
b= input('Enter the second guess: ');
e=input('Enter the tolerance error: ');
n=input('Enter number of iterations: ');
if f(a)*f(b)<0
    for i=1:n

        c=(a*f(b)-b*f(a))/(f(b)-f(a));
        fprintf('c %d = %.4f\n', i,c)
        if abs(f(c))<e
            break
        end
        if f(a)*f(c)<0
            b=c;
        elseif f(b)*f(c)<0
            a=c;
        end
    end
else
    disp('No root between the given brackets.' );
end
```

Output :

```
>> Secant_Method
Enter the equation:
x*sin(x)-1
The equatio is:
f =
     Inline function:
     f(x) = x*sin(x)-1
Enter the first guess:
0
Enter the second guess:
2
Enter the tolerance error:
0.0001
Enter number of iterations:
30
c 1 = 1.0998
c 2 = 1.1212
c 3 = 1.1142
ans =
   1.11416119496263
```

Fixed Point Iteration Method :

```
syms f(x)
f(x) = (cos(x)+1)/3;
f
df = diff(f,x)
x0 = input('Initial guess');
e = input('enter tolerance');
n = input('number of iteration: ');
for i =1: n
    x1=f(x0);
    fprintf('x%d=%.10f\n', i , x1);
    if abs(x1-x0)<e
        break
    end
    x0 = x1;
end
```

Output :

```
>> Fixed_Point_Iteration_Method
f(x) =
cos(x)/3 + 1/3
df(x) =
-sin(x)/3
Initial guess
0
enter tolerance
0.0001
number of iteration:
30
x1=0.6666666667
x2=0.5952957536
x3=0.6093275634
x4=0.6066776832
x5=0.6071822460
x6=0.6070863205
```

Newton Raphson Method :

```
f = @ (x) x*exp(x) - 2;
f
df = @ (x) (x+1)*exp(x) ;
df
x0 = input('Initial guess');
e = input('enter tolerance');
n = input('number of iteration: ');

if (df(x0) ~= 0)
    for i =1:n
        x1 = x0 - (f(x0)/df(x0));
        fprintf('x%d = %.10f\n' , i ,x1)
        if abs(x1 - x0) < e;
            break
        end
        if df(x1) == 0
            disp('failed');
        end
        x0 = x1
    end

else
   disp('failed');
end
```

Output :

```
>> Newton_Raphson_Method
f =
  function_handle with value:
    @(x)x*exp(x)-2
df =
  function_handle with value:
    @(x)(x+1)*exp(x)
Initial guess
3
enter tolerance
0.0001
number of iteration:
30
x1 = 2.2748935342
x0 =
    2.2749
x2 = 1.6430324352
x0 =
    1.6430

x3 = 1.1677272457
x0 =
    1.1677
x4 = 0.9160438366
x0 =
    0.9160
x5 = 0.8555826437
x0 =
    0.8556
x6 = 0.8526123129
x0 =
    0.8526
x7 = 0.8526055020
```

Roots , Polyfit & Interpolation of degree 1:

```
>> %ROOTS :
>> q = [1 0 -4 -9];
>> roots(q)
ans =
   2.7065 + 0.0000i
  -1.3533 + 1.2223i
  -1.3533 - 1.2223i

>>  %POLYFIT :
>>  p = [-3 -2 -1 0 1 2 3];
>>  w = [4.63 2.11 0.67 0.09 0.63 2.15 4.58];
>>  z = polyfit(p,w,2);
>>  poly2sym(z)
ans =
(199*x^2)/400 - (11*x)/2800 + 93/700


>> % INTERPOLATION OF DEGREEE 1 :
>> x = [2 5];
>> y = [1.5 4.0];
>> interp1(x,y,2.5)
ans =
    1.9167
```

MATLAB Code For Least Squares Approximation :

For Straight Line :

```
y = [12 15 21 25];
x = [50 70 100 120];
n = length(y);
i = sum(y);
j = sum(x);
k = sum(y.*x);
l = sum(x.^2);
a = ((l.*i)-(k.*j))/((n.*l)-(j.^2))
b = ((n.*k)-(i.*j))/((n.*l)-(j.^2))
t = [a b]
poly2sym(t)
```

Output :

```
>> Least_Square_for_st_line
a =
    2.2759
b =
    0.1879
t =
    2.2759    0.1879
ans =
(66*x)/29 + 109/580
```

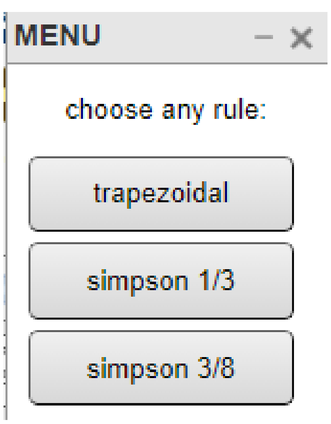
For Parabola :

```
x = [-3 -2 -1 0 1 2 3];
y = [4.63 2.11 0.67 0.09 0.63 2.15 4.58];
n = length(x);
i = sum(x);
j = sum(x.^2);
k = sum(x.^3);
l = sum(x.^4);
m = sum(y);
o = sum(x.*y);
p = sum((x.^2).*y);
```

Trapezoidal , Simpson's $1/3^{rd}$, Simpson's 3/8 Rules:

```
m = menu('choose any rule:','trapezoidal' ,'simpson 1/3 ','simpson 3/8');
% func = @(x) (1./(1 + x.^2));
% i = integral(func,0,6)

x = [0 1 2 3 4 5 6];
y =[1 1/2 1/5 1/10 1/17 1/26 1/37];

h = 1;
n = length(x) - h;

switch m
    case 1
         i = (h/2)*((y(1)+y(7))+2*(y(2)+y(3)+y(4)+y(5)+y(6)));
         disp(i)
    case 2
        j = (h/3)*((y(1)+y(7))+4*(y(2)+y(4)+y(6))+2*(y(3)+y(5)));
        disp(j)
    case 3
        k = ((3*h)/8)*((y(1)+y(7)+3*(y(2)+y(3)+y(5)+y(6))+2*(y(4)+y(7))));
        disp(k)

     otherwise
          disp('try again!')
end
```

Output :




```
>> Trap_simp_Rule
    1.4108
>> Trap_simp_Rule
    1.3662
>> Trap_simp_Rule
    1.3774
```

Newton Forward Difference Interpolation :

```
x = input('Enter list of x: ');
y = input('Enter list of y: ');
p0 = input('Enter point of approximation: ');
n = length(x);
h = x(2) - x(1);
f = zeros(n,n);
f(:,1) = y;
for j=2:n
    for i=j:n
        f(i,j) = f(i,j-1) - f(i-1,j-1);
    end
end
f
c = f(n,n);
for k=n-1:-1:1
    p = poly(x(1))/h;
    p(2) = p(2) - (k-1);
    c = conv(c,p)/k;      % conv= can multiply polynomials.
    m = length(c);
    c(m) = c(m) + f(k,k);

end
z = poly2sym(c)
```

Output :

```
>> Newton_forward_code
Enter list of x:
[0;1;2;3;4;5]
Enter list of y:
[-3;3;11;27;57;107]
Enter point of approximation:
5
f =
    -3     0     0     0     0     0
     3     6     0     0     0     0
    11     8     2     0     0     0
    27    16     8     6     0     0
    57    30    14     6     0     0
   107    50    20     6     0     0
z =
x^3 - 2*x^2 + 7*x - 3
```

Newton Backward Difference Interpolation :

```
x = input('Enter list of x: ');
y = input('Enter list of y: ');
p0 = input('Enter point of approximation: ');
n = length(x);
h = x(2) - x(1);
b = zeros(n,n);
b(:,1) = y;
for j=2:n
    for i=1:n-j+1
        b(i,j) = b(i+1,j-1) - b(i,j-1);
    end
end
b
c = b(1,n);
for k=n-1:-1:1
    p = poly(x(n))/h;
    p(2) = p(2) + (k-1);
    c = conv(c,p)/k;      % conv= can multiply polynomials.
    m = length(c);
    c(m) = c(m) + b(n-k+1,k);

 end
 z = poly2sym(c)
```

Output :

```
>> Newton_Backward_code
Enter list of x:
[0;1;2;3;4;5]
Enter list of y:
[-3;3;11;27;57;107]
Enter point of approximation:
5
b =
    -3     6     2     6     0     0
     3     8     8     6     0     0
    11    16    14     6     0     0
    27    30    20     0     0     0
    57    50     0     0     0     0
   107     0     0     0     0     0
z =
x^3 - 2*x^2 + 7*x - 3
```

Solve , Polyval , Polyint , Ployder:

```
 SOLVE:
 p = sym('x');
 s=3*x^2+6*x-4;
c=(double(solve(s)))
c =
  -0.1667 - 0.7993i
  -0.1667 + 0.7993i

 POLYVAL:
  a = [3 6 -4];
  polyval(a , 2:5)
ans =
    20    41    68   101

 POLYINT:
polyint(a)
ans =
     1     3    -4     0
```

```
poly2sym(a)
ans =
3*x^2 + 6*x - 4
```

```
DIFF:
diff(3*x^2 + 6*x - 4)
ans =
6*x + 6

 DERIVATIVE (POLYDER):
 polyder(a)
ans =
     6     6
```

Methods Of Solving ODEs:

```
m = menu('choose any method:','Euler Method' ,' Improved Euler Method', ..
    'RK-4 Method');
f = input('Enter your function: ');
h = input('Enter your step size: ');
x0 = input('Enter value of x0: ');
y0 = input('Enter value of y0: ');
endpoint = input('Enter how many times loop should iterate: ');
switch m
    case 1
        EulerMethod(f,h,x0,y0,endpoint);
    case 2
        ImprovedEulerMethod(f,h,x0,y0,endpoint);
    case 3
        RKmethod(f,h,x0,y0,endpoint);
end
```

Euler Method:

```
function [x,y] = EulerMethod(f,h,x0,y0,endpoint)
while x0 < endpoint
    y1 = y0+h*f(x0,y0);
    x1 = x0 + h;
    fprintf('x = %.2f, y = %.6f\n', x1, y1);

    % Update x0 and y0 for the next iteration
    x0 = x1;
    y0 = y1;
end
end
```

Output :

```
>> Euler_method(@(x,y)x-y , 0.1,0,1,0.5)
x = 0.10, y = 0.900000
x = 0.20, y = 0.820000
x = 0.30, y = 0.758000
x = 0.40, y = 0.712200
x = 0.50, y = 0.680980
```

Improved Euler Method:

```
function [x,y] = ImprovedEulerMethod(f,h,x0,y0,endpoint)
while x0 < endpoint
    k1 = h * f(x0, y0);
    k2 = h * f(x0 + h, y0 + k1);
    y1 = y0 + (1/2) * (k1 + k2);
    x1 = x0 + h;
    fprintf('x = %.2f, y = %.6f\n', x1, y1);

    % Update x0 and y0 for the next iteration
    x0 = x1;
    y0 = y1;
end
end
```

Output :

```
>> Improved_Euler_method(@(x,y)x-y , 0.1,0,1,0.5)
x = 0.10, y = 0.910000
x = 0.20, y = 0.838050
x = 0.30, y = 0.782435
x = 0.40, y = 0.741604
x = 0.50, y = 0.714152
```

RK-4 Method:

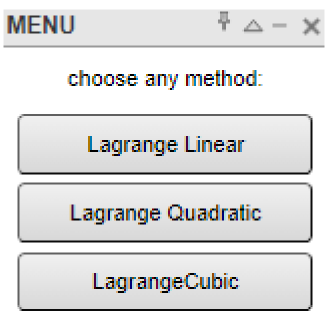
```
function [x,y] = RKmethod(f,h,x0,y0,endpoint)
while x0 < endpoint
    k1 = f(x0, y0);
    k2 = f(x0 + (h / 2), y0 + (k1 / 2));
    k3 = f(x0 + (h / 2), y0 + (k2 / 2));
    k4 = f(x0 + h, y0 + k3);
    y1 = y0 + ((h / 6) * (k1 + 2 * k2 + 2 * k3 + k4));
    x1 = x0 + h;
    fprintf('x = %.2f, y = %.6f\n', x1, y1);
    % Update x0 and y0 for the next iteration
    x0 = x1;
    y0 = y1;
end
end
```

Output :

```
>> RK_4_Method(@(x,y)x-y , 0.1,0,1,0.5)
x = 0.10, y = 0.941250
x = 0.20, y = 0.892422
x = 0.30, y = 0.852896
x = 0.40, y = 0.822090
x = 0.50, y = 0.799459
```

Lagrange's Interpolation:

```
m = menu('choose any method:','Lagrange Linear' ,' Lagrange Quadratic', ...
    'LagrangeCubic');
X = input(' X vector: ');
Y = input('Y vectors: ');
 x = input('x ');
switch m
    case 1
       Lagrange_Linear(X,Y,x);
    case 2
        Lagrange_Quadratic(X,Y,x);
    case 3
        Lagrange_Cubic(X,Y,x);
end
```



Lagrange's Linear Interpolation:

```
function [g] = Lagrange_Linear(X,Y,x)
a = (((x-X(2))/(X(1)-X(2)))*Y(1));
b = (((x-X(1))/(X(2)-X(1)))*Y(2));
g = a+b
end
```

Output :

```
>> lagranges_main
 X vector:
[2 5]
Y vectors:
[1.5 4.0]
x
3

g =

    2.3333
```

Lagrange's Quadratic Interpolation:

```
function [g] = Lagrange_Quadratic(X,Y,x)
%syms x;
a = ((((x-X(2))*(x-X(3)))/((X(1)-X(2))*(X(1)-X(3))))*Y(1));
b = ((((x-X(1))*(x-X(3)))/((X(2)-X(1))*(X(2)-X(3))))*Y(2));
c = ((((x-X(1))*(x-X(2)))/((X(3)-X(1))*(X(3)-X(2))))*Y(3));
g = a+b+c
end
```

Output :

```
>> lagranges_main
 X vector:
[3 4 5 ]
Y vectors:
[1 2 4]
x
3

g =

     1
```

Lagrange's Cubic Interpolation:

```
function [g] = Lagrange_Cubic(X,Y,x)
v1 = ((x-X(2))*(x-X(3))*(x-X(4)))/((X(1)-X(2))*(X(1)-X(3))*(X(1)-X(4)));
v2 = ((x-X(1))*(x-X(3))*(x-X(4)))/((X(2)-X(1))*(X(2)-X(3))*(X(2)-X(4)));
v3 = ((x-X(1))*(x-X(2))*(x-X(4)))/((X(3)-X(1))*(X(3)-X(2))*(X(3)-X(4)));
v4 = ((x-X(1))*(x-X(2))*(x-X(3)))/((X(4)-X(1))*(X(4)-X(2))*(X(4)-X(3)));
g = (v1*Y(1))+(v2*Y(2))+(v3*Y(3))+(v4*Y(4))
end
```

Output :

```
>> lagranges_main
 X vector:
[0.4 0.5 0.7 0.8]
Y vectors:
[-0.916291 -0.693147 -0.356675 -0.223144]
x
0.605

g =

   -0.5017
```

Jacobi Method :

```
function [] = Jacobi_Method(a,b,tol,x,y,z)
if ((abs(a(1,1)))>=(abs(a(1,2))+abs(a(1,3))) && (abs(a(2,2)))>=(abs ...
        (a(2,1))+abs(a(2,3))) && (abs(a(3,3)))>=(abs(a(3,1))+abs(a(3,2))))
    fprintf('k      x1(k)      x2(k)      x3(k) \n');
    for i=0:100
        A = [i; x; y; z];
        fprintf('%d    %9.7f    %9.7f    %9.7f \n', A);
        x1=(1/a(1,1))*(b(1)-(a(1,2)*y)-(a(1,3)*z));
        x2=(1/a(2,2))*(b(2)-(a(2,1)*x)-(a(2,3)*z));
        x3=(1/a(3,3))*(b(3)-(a(3,1)*x)-(a(3,2)*y));
        if ((abs(x-x1)<=tol) && (abs(y-x2)<=tol) && (abs(z-x3)<=tol))
            break;
        end
        x=x1;
        y=x2;
        z=x3;
    end
    else
        disp("Your value may diverge")
    end
end
```

Output :

```
>> Jacobi_Method([10 3 1; 3 10 2; 1 2 10],[19; 29; 35],0.0001,0,0,0)
k      x1(k)      x2(k)      x3(k)
0    0.0000000    0.0000000    0.0000000
1    1.9000000    2.9000000    3.5000000
2    0.6800000    1.6300000    2.7300000
3    1.1380000    2.1500000    3.1060000
4    0.9444000    1.9374000    2.9562000
5    1.0231600    2.0254400    3.0180800
6    0.9905600    1.9894360    2.9925960
7    1.0039096    2.0043128    3.0030568
8    0.9984005    1.9982158    2.9987465
9    1.0006606    2.0007306    3.0005168
10    0.9997292    1.9996985    2.9997878
11    1.0001117    2.0001237    3.0000874
12    0.9999542    1.9999490    2.9999641
```

Gauss-Seidel Method :

```
function [] = Gauss_Seidel_Method(a,b,tol,x,y,z)
if(abs(a(1,1)))>=(abs(a(1,2))+(abs(a(1,3)))) && (abs(a(2,2)))>=(abs(a ...
        (2,1))+(abs(a(2,3)))) && (abs(a(3,3)))>=(abs(a(3,1))+(abs(a(3,2))))
    fprintf(' k      x1(k)      x2(k)      x3(k) \n');
    for i=0:100
        A=[i;x;y;z];
        fprintf(' %d    %9.7f    %9.7f    %9.7f \n',A);
        x1=(1/a(1,1))*(b(1)-(a(1,2)*y)-(a(1,3)*z));
        x2=(1/a(2,2))*(b(2)-(a(2,1)*x)-(a(2,3)*z));
        x3=(1/a(3,3))*(b(3)-(a(3,1)*x)-(a(3,2)*y));
        if ((abs(x-x1)<=tol) && (abs(y-x2)<=tol) && (abs(z-x3)<=tol))
            break;
        end
        x=x1;
        y=x2;
        z=x3;
    end
    else
        disp("your value may diverge");
    end
end
```

Output :

```
>> Gauss_Seidel_Method([10 3 1; 3 10 2; 1 2 10],[19; 29; 35],0.0001,0,0,0)
 k      x1(k)      x2(k)      x3(k)
 0    0.0000000    0.0000000    0.0000000
 1    1.9000000    2.9000000    3.5000000
 2    0.6800000    1.6300000    2.7300000
 3    1.1380000    2.1500000    3.1060000
 4    0.9444000    1.9374000    2.9562000
 5    1.0231600    2.0254400    3.0180800
 6    0.9905600    1.9894360    2.9925960
 7    1.0039096    2.0043128    3.0030568
 8    0.9984005    1.9982158    2.9987465
 9    1.0006606    2.0007306    3.0005168
 10    0.9997292    1.9996985    2.9997878
 11    1.0001117    2.0001237    3.0000874
 12    0.9999542    1.9999490    2.9999641
```

Eign & lu Command :

```
% eig Command (for Eign Vector(v) & diagonal matrix whose
% diagonal entries are Eign Values(d)).
 B = [1 -1 0; -1 2 1; 0 1 1];
 [v,d] = eig(B)
 v =
     0.5774    0.7071   -0.4082
     0.5774   -0.0000    0.8165
    -0.5774    0.7071    0.4082
 d =
     0.0000         0         0
          0    1.0000         0
          0         0    3.0000
```

```
% lu Command
 C = [2 1 3; 4 3 10; 2 4 17];
 [L,U] = lu(C)
 L =
     0.5000   -0.2000    1.0000
     1.0000         0         0
     0.5000    1.0000         0
 U =
     4.0000    3.0000   10.0000
          0    2.5000   12.0000
          0         0    0.4000
```

Power Method :

By Using Rayleigh Approach :

```
function [] = Rayleigh_Power_m(A,x0,tol)

for i=0:7
    x = A*x0;
    j = min(x);
    x = x/j;
    if(abs(x-x0)<tol)
        break;
    end
    x0 = x;
    lamda = ((A*x0).*x0)/(x0.*x0);
end
fprintf('The eign vector is: \n');
x
fprintf('The eign value is: %.4f \n', lamda);
end
```

Output :

```
>> Rayleigh_Power_m([2 -12; 1 -5],[1;1],0.0001)
The eign vector is:

x =

    1.0000
    0.3336

The eign value is: -2.0035
The eign value is: -0.2229
The eign value is: 0.0000
The eign value is: 0.0000
```

By Using Scaling :

```
function [] = Scaling_Power_m(A,x0,tol)

for i=0:7
    x = A*x0;
    j = max(x);
    x = x/j;
    if(abs(x-x0)<tol)
        break;
    end
    x0 = x;
    lamda = ((A*x0).*x0)/(x0.*x0);
end
fprintf('The eign vector is: \n');
x
fprintf('The eign value is: %.4f \n', lamda);
end
```

Output :

```
>> Scaling_Power_m([1 2 0; -2 1 2; 1 3 1],[1;1;1],0.0001)
The eign vector is:

x =

    0.5001
    0.5001
    1.0000

The eign value is: 0.0000
The eign value is: 0.0000
The eign value is: 0.0000
The eign value is: 0.0000
The eign value is: 0.0000
The eign value is: 0.0000
The eign value is: 0.7502
The eign value is: 0.7501
The eign value is: 3.0003
```