```
%Bisection Method to find a real root
f= @(x) x^3-4*x-9;
a = input("Enter first initial guess:\n");
b = input("Enter second initial guess:\n");
tol = 0.00001;
n = 1;
if f(a)*f(b)>0
    fprintf("Error: wrong initial guesses");
elseif f(a)*f(b)<0
    while abs(b-a)>tol
        x = (a+b)/2;
        fprintf("n: %d\t a:%8.5f\t b:%8.5f\t f(a):%8.5f\t f(b):%8.5f\t x:%8.5f\t
Error:%8.5f\n",n,a,b,f(a),f(b),x,abs(f(x)));
        if f(a)*f(x)<0
            b=x;
        elseif f(b)*f(x)<0
            a=x;
        end
        n=n+1;
    end

end
```

```
% Fitting quadratic curve y = a + bx + cx^2
X = input("Enter the values of X:\n");
Y = input("Enter the values of Y:\n");
n = length(X);

SX = sum(X);
SXX = sum(X.*X);
SXXX = sum(X.*X.*X);
SXXXX = sum(X.*X.*X.*X);
SY = sum(Y);
SXY = sum(X.*Y);
SXXY = sum(X.*X.*Y);

% Constructing the matrices for the normal equations
A = [n SX SXX; SX SXX SXXX; SXX SXXX SXXXX];
B = [SY; SXY; SXXY];

% Solving the normal equations
Z = A\B;

% Displaying the results
fprintf("Required quadratic curve: y = %0.4f + %0.4fx + %0.4fx^2\n", Z(1), Z(2),
Z(3));

% Plotting the data and best fit
f = @(x) Z(1) + Z(2)*x + Z(3)*x.^2;
plot(X, Y, '*r');
hold on;
fplot(f, [X(1),X(n)]);
hold off;
```

```matlab
%Rk=4 method
%example dy/dx=f(x,y,z)=xz+1;y(3)=2
f=@(x,y,z) x*z+1;
g=@(x,y,z) -x*y;
%define the initial conditions y(3)=2;
x0=0;
y0=0;
z0=1;
%define the step size and the number of steps
xn=12;
n=10;
h=(xn-x0)/n;
%initialize array to store the solution
x=zeros(1,n+1);
y=zeros(1,n+1);
%set the initial condition in the array
x(1)=x0;
y(1)=y0;
z(1)=z0;
%implement the eulers method
for i=1:n
    L1=h*g(x(i),y(i),z(i));
      L2=h*g(x(i)+h/2,y(i)+k1/2,z(i)+L1/2);
      L3=h*g(x(i)+h/2,y(i)+k2/2,z(i)+L2/2);
      L4=h*g(x(i)+h,y(i)+k3,z(i)+L3);
      z(i+1)=z(i)+(L1+2*L2+2*L3+L4)/6;
      x(i+1)=x(i)+h;
      k1=h*f(x(i),y(i),z(i));
      k2=h*f(x(i)+h/2,y(i)+k1/2,z(i)+L1/2);
      k3=h*f(x(i)+h/2,y(i)+k2/2,z(i)+L2/2);
      k4=h*f(x(i)+h,y(i)+k3,z(i)+L3);
      y(i+1)=y(i)+(k1+2*k2+2*k3+k4)/6;
end
%solution table
T=table(x',y',z','VariableNames',{'xn','yn','zn'});
disp(T);
%plot the approximation solution
plot(x,y,'r--o');
hold on;
%plot the exact solution
exact_solution=@(x)-exp(-x)+exp(-3)+2;
fplot(exact_solution,[3,12],'-b');

xlabel('x');
ylabel('y');
title(' RK-4 second order differential Method');
legend('Approximation solution ','exact solution');
grid on;
hold off;
```

```
%eulers method
f = @(x,y) exp(-x);

xo = 3;
yo = 2;

xn = 12;
n = 100;
h = (xn-xo)/n;

x = zeros(1, n+1);
y = zeros(1, n+1);

x(1) = xo;
y(1) = yo;

%Implement Euler's method
for i = 1:n
    x(i+1) = x(i)+h;
    y(i+1) = y(i)+h*f(x(i),y(i));
end

%Solution table
T = table(x', y','VariableNames',{'xn','yn'});
disp(T)

%Plot the approximation solution
plot(x,y,'r--o');
hold on;

%Plot the exact solution
exact_solution = @(x) -exp(-x)+exp(-3)+2;
fplot(exact_solution,[3,12],'-b');

xlabel('x');
ylabel('y');
title('Eulers methods');
legend('Approximation solution','exact_solution');
grid on;
hold off;
```

```
%False position Method to find a real root
f= @(x) log10(x)-cos(x);
a = input("Enter first initial guess:\n");
b = input("Enter second initial guess:\n");
tol = 0.00001;
n = 1;
if f(a)*f(b)>0
    fprintf("Error: wrong initial guesses");
elseif f(a)*f(b)<0
    while abs(b-a)>tol
        x = (a*f(b)-b*f(a))/(f(b)-f(a));
        fprintf("n: %d\t a:%8.5f\t b:%8.5f\t f(a):%8.5f\t f(b):%8.5f\t x:%8.5f\t
Error:%8.5f\n",n,a,b,f(a),f(b),x,abs(f(x)));
        if f(a)*f(x)<0
            b=x;
        elseif f(b)*f(x)<0
            a=x;
        end
        n=n+1;
    end

end
```

```
%Gauss elimination method
A = [1 3 -2 ; 3 5 6; 2 4 3];
B = [5; 7; 8];
n = size(A,1);
x = zeros(n,1);

for j = 1:n-1
    for i = j+1:n
        m = (A(i,j)/A(j,j));
        A(i,:) = A(i,:)-m*A(j,:);
        B(i,:) = B(i,:)-m*B(j,:);
    end
end

x(n,:) = B(n,:)/A(n,n);
for i = n-1:-1:1
    x(i,:) = (B(i,:)-A(i,i+1:n)*x(i+1:n,:))/A(i,i);
end

disp('Row Echelon form of the augemented matrix');
disp(A);
disp('Solution vector x:');
disp(x);
disp('Direct Method:');
disp(A\B);


%Gauss jordan
A = [1 3 -2 ; 3 5 6; 2 4 3];
B = [5; 7; 8];
n = size(A,1);
x = zeros(n,1);

%form the augmented matrix
Aug = [A B];

%forward elimination and making the diagonal elements 1
for j = 1:n
    Aug(j,:) = Aug(j,:) / Aug(j,j);
    for i = 1:n
        if i~=j
            m = Aug(i,j);
            Aug(i,:) = Aug(i,:)-m*Aug(j,:);
        end
    end
end

%the last column of Aug now contains the solution
x = Aug(:, end);

disp('Reduced row echelon form of the augemented matrix');
disp(Aug);
disp('Solution vector x:');
disp(x);
```

```matlab
%Gauss seidal
A = [20 1 -2; 3 20 -1; 2 -3 20];
B = [17; -18; 25];
n = length(B);
x = zeros(n,1);%Initial guess
tol = 1e-4;%Tolerance
N = 100;%Maximum number of iterations

%Gauss-Seidal iteration
for k = 1:N
    x_old = x;
    fprintf("Iteration: %d \n", k);
    for i = 1:n
        Sum = 0;
        for j = 1:n
            if j~=i
                Sum = Sum + A(i,j) *x(j);
            end
        end
        x(i) = (B(i)-Sum)/A(i,i);
        fprintf("X(%d): %f \n", i, x(i));
    end

    %check for convergence
    if norm(x-x_old, inf)<tol
        break;
    end
end
disp('Exact Solution:');
disp(A\B);
disp('Approx. Solution vector x:');
disp(x);
```

```matlab
%Inverse of matrix
A = [1 3 -2 ; 3 5 6; 2 4 3];
n = size(A,1);
x = zeros(n,1);

%form the augmented matrix
Aug = [A eye(n)];

%forward elimination and making the diagonal elements 1
for j = 1:n
    Aug(j,:) = Aug(j,:) / Aug(j,j);
    for i = 1:n
        if i~=j
            m = Aug(i,j);
            Aug(i,:) = Aug(i,:)-m*Aug(j,:);
        end
    end
end

%Extract the inverse of the matrix from the augmented matrix
A_inv = Aug(:,n+1:end);

%the last column of Aug now contains the solution
x = Aug(:, end);

disp('Reduced row echelon form of the augemented matrix');
disp(Aug);
disp('Inverse of Matrix A:');
disp(A_inv);
```

```matlab
%lagrange's interpolation method
xn = input("Values of x:\n");
yn = input("Values of y:\n");
x = input("Enter value of x at which value of y is required:\n");
n = length(xn);
y = 0;
for i = 1:n
    nr = 1;
    dr = 1;
    for j=1:n
        if i~=j
            nr=nr*(x-xn(j));
            dr=dr*(xn(i)-xn(j));
        end
    end
    y=y+(nr/dr)*yn(i);
end
fprintf("f(%.2f)=%.2f\n",x,y);
```

```
%Modified eulers method
f = @(x,y) exp(-x);

xo = 3;
yo = 2;

xn = 12;
n = 100;
h = (xn-xo)/n;

x = zeros(1, n+1);
y = zeros(1, n+1);

x(1) = xo;
y(1) = yo;

%Implement Euler's method
for i = 1:n
    x(i+1) = x(i)+h;
    k1 = h*f(x(i),y(i));
    k2 = h*f(x(i)+h, y(i)+k1);
    y(i+1) = y(i)+(k1+k2)/2;
end

%Solution table
T = table(x', y','VariableNames',{'xn','yn'});
disp(T)

%Plot the approximation solution
plot(x,y,'r--o');
hold on;

%Plot the exact solution
exact_solution = @(x) -exp(-x)+exp(-3)+2;
fplot(exact_solution,[3,12],'-b');

xlabel('x');
ylabel('y');
title('Eulers methods');
legend('Approximation solution','exact_solution');
grid on;
hold off;
```

```matlab
%Newton's backward differentiation
xn = input("Values of x:\n");
yn = input("Values of y:\n");
n = length(xn);
h = xn(2) - xn(1);

% Computing backward difference table:
D = zeros(n, n);
D(:, 1) = yn;
for j = 2:n
    for i = n:-1:j
        D(i, j) = D(i, j-1) - D(i-1, j-1);
    end
end
disp(D);

%computing first derivative at x0
y1 = 0;
for i = 2:n
    y1 = y1 +D(n,i)/(i-1);
end
y1 = y1/h;
fprintf("The first derivative at %.2f = %.2f", xn(7), y1);

%computing second derivate at x0:
y2 = 1/h^2*(D(n,3)+D(n,4)+11/12*D(n,5)+5/6*D(n,6)+137/180*D(n,7));
fprintf("Second derivate at %.2f =%.2f\n", xn(7),y2);
```

```
%Newtons forward difference
xn = input("Values of x:\n");
yn = input("Values of y:\n");
x = input("Enter value of x at which value of y is required:\n");
n = length(xn);
h=xn(2)-xn(1);
p=(x-xn(1))/h;

%Computing forward difference table:
D = zeros(n,n);
D(:,1)=yn;
for j=2:n
    for i=j:n
        D(i,j)=(D(i,j-1)-D(i-1,j-1));
    end
end
disp(D);

y=yn(1);
for i=2:n
    s=1;
    for j=1:i-1
        s=s*(p-j+1);
    end
    y=y+(s*D(i,i))/factorial(i-1);%Represent diagonal element value D(i,i)
end
fprintf("f(%.2f)=%.2f\n", x,y);
```

```
%newtons divided difference
xn = input("Values of x:\n");
yn = input("Values of y:\n");
x = input("Enter value of x at which value of y is required:\n");
n = length(xn);
y = yn(1);

%Computing divided difference table:
D = zeros(n,n)
D(:,1)=yn;
for j=2:n
    for i=j:n
        D(i,j)=(D(i,j-1)-D(i-1,j-1))/(xn(i)-xn(i-j+1));
    end
end
disp(D);

%computing the formula:
y=yn(1);
for i=2:n
    p=1;
    for j=1:i-1
        p=p*(x-xn(j));
    end
    y=y+p*D(i,i);%Represent diagonal element value D(i,i)
end
fprintf("f(%.2f)=%.2f\n", x,y);
```

```
%newtons forward differentiation
xn = input("Values of x:\n");
yn = input("Values of y:\n");
n  = length(xn);
h = xn(2)-xn(1);

%Computing forward difference table:
D = zeros(n,n);
D(:,1)=yn;
for j=2:n
    for i=j:n
        D(i,j)=(D(i,j-1)-D(i-1,j-1));
    end
end
disp(D);

%computing first derivative at x0
y1 = 0;
for i = 2:n
    y1 = y1 + (-1)^i * D(i,i)/(i-1);%purai data rakhna cha vane D(i,i) ko thau ma
D(i+1) garna parxa
end
y1 = y1/h;
fprintf("The first derivative at %.2f = %.2f", xn(1), y1);

%computing second derivate at x0:
y2 = 1/h^2*(D(3,3)-D(4,4)+11/12*D(5,5));
fprintf("Second derivate at %.2f =%.2f\n", xn(1),y2);
```

```matlab
%newtons raphson method
f = @(x) x^3-9;
df = @(x) 3*x^2;
a = input("Enter first initial guess:\n");
tol = 0.00001;
n = 1;


while abs(f(a)) > tol
    x = a - f(a)/ df(a);
    a = x;
    fprintf("iteration %d: %8.5f\n", n, x);
    n = n + 1;
end
```

```matlab
%Power method
%Define matrix A
A = [2 -2 4; 2 3 2; -1 1 1];

%set tolerance and maximum number of iterations
tol = 1e-4;
N = 100;

%Get the size of the matrix A
n = size(A, 1);

%Initialize the vector x with an initial guess
X = [1; 0; 0];
k = 1;

%Perform the power method
for i=1:N
    X_old = X;
    k_old = k;
    Y = A*X;

    %Estimate the largest eigen value
    k = max(abs(Y));

    %Normalize the vector Y
    X = Y/k;

    %Check for convergence
    if norm(X_old-X)<tol && norm(k_old-k)<tol
        break;
    end
end

%Display the results
fprintf('Largest eigenvalue: %4f\n',k);
disp('Corresponding eigen vector:');
disp(X);
```

```
%Romberg method
h=1;
I1=trap(h);
I2=trap(h/2);
I3=trap(h/4);

R1=1/3*(4*I2-I1);
R2=1/3*(4*I3-I2);
R=1/3*(4*R2-R1);

disp(R1);
disp(R2);
disp(R);
```

```
%Secant method
f = @(x) x^2 - 3*x + 2;
a = input("Enter first initial guess:\n");
b = input("Enter second initial guess:\n");
tol = 0.00001;
n = 1;


       while abs(b - a) > tol
        x = (a * f(b) - b * f(a)) / (f(b) - f(a));
        a = b;
        b = x;
        fprintf("iteration %d: %8.5f\n", n,x);
        n = n + 1;
    end
```

```matlab
%Simpson's 1/3 rule
y=@(x) 1/(1+x^2);

a=0;
b=6;
n=6;
h=(b-a)/n;
x_values=a:h:b;
y_values=arrayfun(y,x_values);

%table
T=table([x_values]',[y_values]','VariableNames',{'x','y'});
disp(T);

%formula
s=y(a)+y(b);

%for odd
for i=1:2:n-1
   s=s+4*y(a+i*h);
end

for i=2:2:n-2
   s=s+2*y(a+i*h);
end

I=h/3*s;
fprintf("Approximated intregal value(I)=%.4f\n",I);
```

```
%simpson's 3/8

y=@(x) 1/(1+x^2);

a=0;
b=6;
n=6;
h=(b-a)/n;
x_values=a:h:b;
y_values=arrayfun(y,x_values);
%table
T=table([x_values]',[y_values]','VariableNames',{'x','y'});
disp(T);

%formula
s=y(a)+y(b);

%for odd
for i=1:3:n-2
  s=s+3*(y(a+i*h)+y(a+(i+1)*h));
end

for i=3:3:n-3
   s=s+2*y(a+i*h);
end

I=3/8*h*s;
fprintf("Approximated intregal value(I)=%.4f\n",I);
```

```matlab
%simulataneous diff equation
f = @(x,y,z) x*z+1
g = @(x,y,z) -x*y

xo = 0;
yo = 0;
zo = 1;

xn = 0.2;
n = 10;
h = (xn-xo)/n;

x = zeros(1, n+1);
y = zeros(1, n+1);
z = zeros(1,n+1);

x(1) = xo;
y(1) = yo;
z(1) = zo;

for i = 1:n
    x(i+1) = x(i)+h;

    l1 = h*g(x(i),y(i),z(i));
    k1 = h*f(x(i),y(i),z(i));

    l2 = h*g(x(i)+h/2,y(i)+k1/2, z(i)+l1/2);
    k2 = h*f(x(i)+h/2, y(i)+k1/2,z(i)+l1/2);

    l3 = h*g(x(i)+h/2, y(i)+k2/2, z(i)+l2/2);
    k3 = h*f(x(i)+h/2,y(i)+k2/2, z(i)+l2/2);

    l4 = h*g(x(i)+h, y(i)+k3, z(i)+l3);
    k4 = h*f(x(i)+h,y(i)+k3, z(i)+l3);

    y(i+1) = y(i)+(k1+2*k2+2*k3+k4)/6;
    z(i+1) = z(i)+(l1+2*l2+2*l3+l4)/6
end

%Solution table
T = table(x', y','VariableNames',{'xn','yn'});
disp(T)

%Plot the approximation solution
plot(x,y,'r--o');
hold on;
plot(x,z,'b--o');

xlabel('x');
ylabel('y');
title('Eulers methods');
grid on;
hold off;
```

```
%trapezoidal rule
%function define this is the solution of the romberge
y=@(x) 1/(1+x^2);

a=0;
b=6;
n=6;
h=(b-a)/n;
x_values=a:h:b;
y_values=arrayfun(y,x_values);

%table
T=table([x_values]',[y_values]','VariableNames',{'x','y'});
disp(T);

%formula
s=y(a)+y(b);
for i=1:n-1
   s=s+2*y(a+i*h);
end
I=h/2*s;
fprintf("Approximated intregal value(I)=%.4f\n",I);
```