1. Bisection Method

Aim: The bisection method is used to find the roots of a polynomial equation. It separates the interval and subdivides the interval in which the root of equation lies. The principle behind this method is the intermediate theorem for continuous functions.

Pseudocode:

```
clc
f = input('Enter non-linear equations :');
a = input(' Enter first guess :');
b = input(' Enter Second guess:');
n = input(' Number of iteration :');
if f(a) * f(b) < 0
for i = 1 : n
c = (a+b) / 2
if f(a) * f(c) < 0
b = c ;
```

```
else if f(b)*f(c) < 0
a = c;
end
end
else
disp('No root beetween given brackets')
end
```

[Input]

```
f = 2^x - 5x+2;
a = 0;
b = 1;
n = 20;
```

[Output]

2: Experiment Name: False Position Method

Aim: An algorithm for finding roots which retains that prior estimate for which the function value has opposite sign from the function value at the current best estimate of the root. In this way, the method of false position keeps the root bracked

Pseudocode:

f = input(' Enter non-linear equations:')

a = input(' Enter first guess:');

b = input(' Enter second guess:');

if f(a) * f(b) < 0

for i = 1:n

$$c = \frac{a(f(b)) - b(f(a))}{f(b) - f(a)}$$

v

```
if f(a) * f(c) < 0
b = c;
else if f(b) * f(c) < 0
a = c;
end
end
else
disp('No root between given brackets');
end
```

[input]

```
f = cos(x) - x;
a = 0.5
b = 1
n = 20
```

[output]

Method

3: Experiment Name: Newton Raphson Method

Aim: The Newton - Raphson method (also known as Newton's method) is a way to quickly find a good approximation for the root of a real valued function $f(x) = 0$. It uses the idea that a continuous and differentiable function can be approximated by a straight line tangent to it.

Pseudocode:

$f = x^2 + 2x + 3;$

$df = 2x + 2;$

$e = 10^4;$

$x_0 = 0;$

$n = 10;$

```
if df(x0)~=0
for i=1:n
x1=x0-(f(x0)/df(x0))
fprintf('x%d=%.4f\n', i,x1)
if abs(x1-x0)<e
break
end
x0=x1;
end
else
disp('Newton Raphson Failed');
end
```

Method:

3. Experiment Name: Secant Method

Aim: In numerical analysis, the secant method is a root finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function f. The secant method can be thought of as a finite-difference approximation of Newton's method.

Pseudocode:

```
clc;
f = @(x) (x^2-2);
x0 = input('Enter x0: ');
x1 = input('Enter x1: ');
tol = input('Enter tolarance: ');
itr = input(' Enter Iteration: ');
P = 0;
for i = 1: itr
```

```
x2=(x0*f(x1)-x1*f(x0))/(f(x1)-f(x0));
if abs(x2-x1)<tol
p=1;
k=i;
break;
else
x0=x1;
x1=x2;
end
end
if p==1
fprintf('Solution is %f at iteration %i', x2,k)
else
fprintf('No convergent solution exist in the given number iteration')
end
```

[Input]

x0=0.5

x1=5

tolarance =0.0001

iteration =10

Method

5. Experiment Name: Fixed Point Method

Aim: Fixed point method allows us to solve nonlinear one variable equations. We build an iterative method, using a sequence which converges to a fixed point of g(x), this fixed point is the exact solution of f(x)=0.

Pseudocode:

```
clc
g = @ (x) (2^x+2) /5 ;
x0 = input('Enter initial values:');
e=0.0001;
n=input('Enter Iteration:');

for i = 1:n
x1 =g(x0) ;
if abs(x1-x0) <e
fprintf ('x%d =%4.f\n', i,xi).

break
end   x0=x1;  end
```

Method

6: Experiment Name: Basic Gauss Eliminati
Method

Aim: The goal of Gaussian elimination is to get the matrix in row echelon form. If a matrix is in row echelon form, that means that reading from left to right each row will start with at least one more zero term than the row above it.

Pseudocode:

```
A = input('Enter your coefficient
                         matrix:');
B = input('Enter source vector:');
N = length(B);
X = zeros(N,1);
Aug = [A B]
for j = 1:N-1
```

```
for i=j+1:N
    m=Aug(i,j)/Aug(j,j);
    Aug(i,:)=Aug(i,:)-m*Aug(j,:);
end
end
Aug
x(N)=Aug(N,N+1)/Aug(N,N);
for k=N-1:1:1
  x(k)=(Aug(k,N+1)-Aug(k,k+1:N)*
                     x(k+1:N))/Aug(k,k);
end
x
```

Method

7. Experiment Name: Jacobi Iteration Method

Aim: In Neumerical linear algebra, the jacobi method is an iterative algorithm for determining the solutions of a strictly diagonally dominant system of linear equations. Each diagonal element is solved for, and an approximate value is plugged in, The pocess is then iterated until it converges.

Pseudocode:

```
A = input(' Enter co-efficient matrix A:');
B = input(' Enter source vector B: ')
P = input(' Enter initial guess vector: ');
n = input(' Enter no of Iterations: ');
e = input(' Enter Tolerance: ');
N = length(B);
X = zeros(N,1);
```

```
for j = i : n
    for i = 1 : N
    x(i) = (B(i)/A(i,i)) - (A(i,
            [1:i-1, i+1:N]) x P([1:i-1, i
                    +1:N])) /A(i,i);

    end
    fprintf('Iteration no %d \n', j)

    x
    if abs(x - P) < e
    break
    end
    p - x;
    end;
```