LAB - 1

To implement bisection, secant, and Newton-Raphson method.

OBJECTIVE:

To implement and analyze three numerical methods for finding the roots of a given function: The Bisection Method, the Secant Method, and the Newton-Raphson Method

THEORY:

1. Bisection Method:

The Bisection Method is a numerical approach for finding roots of a function by systematically reducing an interval where the function changes sign. It is a type of bracketing method, meaning that it confines the root within a shrinking interval. The method relies on the property that a continuous function, f(x)f(x)f(x), will have at least one root in the interval [a,b][a, b][a,b] if f(a)f(a)f(a) and f(b)f(b)f(b) have opposite signs.

Algorithm:

i. Choose an interval [a, b] such that f(a) and f(b) have opposite signs.
ii. Compute the midpoint c = (a + b) / 2.
iii. If f(c) is sufficiently close to zero, return c as the root.
iv. Otherwise, replace either a or b with c based on the sign of f(c), and repeat.

2. Secant Method:

The Secant Method is an iterative numerical technique for finding the root of a function by approximating the derivative using a finite difference. Unlike the Bisection Method, it does not require the initial interval to enclose a root or for the function to change signs within an interval. Instead, it uses two initial approximations and updates them iteratively based on the secant line.

Algorithm:

i. Start with two initial guesses, x0 and x1.
ii. Compute the next approximation using the formula.

$$x_2 = x_1 - f(x_1).\frac{x_1 - x_0}{f(x_1) - f(x_0)}$$

iii. Repeat the process until convergence is achieved.

3. Newton-Raphson Method:

The Newton-Raphson Method is an iterative numerical technique for finding the root of a function by leveraging its first derivative to refine approximations. It starts with an initial guess and repeatedly improves the estimate using the function's slope.

Algorithm:

i. Start with an initial guess, x0.
ii. Compute the next approximation using the formula.

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

iii. Repeat until convergence is achieved.

PROGRAM DEMONSTRATION:

Source Code:

Bisection method

//Example : f(x) = x^2 -5x-10

```
#include<stdio.h>
#include<math.h>
float evaluate(float x);
float bisect(float x1,float x2);

int main(){
   int ni, i;
   float a,b,m;
   printf("Enter Maximum no of Iterations: ");
   scanf("%d",&ni);

//.....Compute x1 and x2.....

  do{
      printf("Enter the value of a and b (starting boundary): ");
      scanf("%f%f",&a,&b);
      if(evaluate(a)*evaluate(b)>0){
        printf("\nRoots are Invalid\n");
        continue;
      }else{
         printf("\nRoots Lie between %f and %f\n",a,b);
         break;
      }
   } while(1);

 //.....Find root.....

         printf("\nIterations\ta\t\tb\t\troots\t\tfunctional value\n");

   while(i<=ni){
     m = bisect(a,b);   //find the mid point
     if(evaluate(m)*evaluate(a)<0)
       b=m;  //x2 is shifted
     else
      a=m;
     printf("      %d   \t%f\t%f\t%f\t%f\n",i,a,b,m, evaluate(m));

     i++;
   }
   printf("\nRoot=%f  , functinal value=%f Total Iterations=%d",m, evaluate(m),--i);
   return 0;
}

float evaluate(float x){
  return x*x - 5*x - 10;
}

float bisect(float x1,float x2){
   return (x1+x2)/2;
}
```

Output:

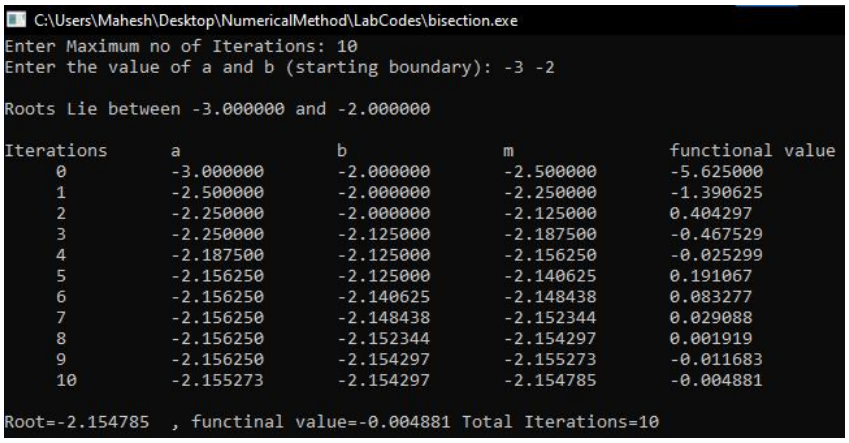
```
C:\Users\Mahesh\Desktop\NumericalMethod\LabCodes\bisection.exe
Enter Maximum no of Iterations: 10
Enter the value of a and b (starting boundary): 6.5
7

Roots Lie between 6.500000 and 7.000000

Iterations      a               b               roots           functional value
     0          6.500000        6.750000        6.750000        1.812500
     1          6.500000        6.625000        6.625000        0.765625
     2          6.500000        6.562500        6.562500        0.253906
     3          6.500000        6.531250        6.531250        0.000977
     4          6.515625        6.531250        6.515625        -0.124756
     5          6.523438        6.531250        6.523438        -0.061951
     6          6.527344        6.531250        6.527344        -0.030502
     7          6.529297        6.531250        6.529297        -0.014767
     8          6.530273        6.531250        6.530273        -0.006897
     9          6.530762        6.531250        6.530762        -0.002960
     10         6.531006        6.531250        6.531006        -0.000992

Root=6.531006  , functinal value=-0.000992 Total Iterations=10
```

//Example : f(x) = x^3+10

Only need to change:

```
float evaluate(float x)
{
  return x*x*x + 10;
}
```

Output:



Secant Method

//Example : f(x) = $e^x - x^2 + 3x - 2$

```
#include<stdio.h>
#include<math.h>
//to find answers till 4 correct decmal places
#define e 0.00001

float f(float x);
float evaluate(float x1,float x2);

int main()
{
   int ni,i=1;
   float x1,x2,x;

   printf("Enter Maximum no of Iterations: ");
   scanf("%d",&ni);

//.....Compute x1 and x2 to check for the sign change

  do{
      printf("Enter initial values of x1 and x2: ");
      scanf("%f%f",&x1,&x2);
      printf("\nf(x1)= %f \nf(x2)= %f\n", f(x1),f(x2));

      if(f(x1)*f(x2)>0)
      {
        printf("Roots are Invalid\n");
        continue;
      }
      else
      {
         printf("Roots Lie between %f and %f\n\n",x1,x2);
         break;
      }
   } while(1);

   x = evaluate(x1,x2);
   do{
     x1=x2;
     x2=x;
     printf("Iterations=%d  Root=%f\n",i,x);
     x = evaluate(x1,x2);
     if(fabs(x-x2)<e){
       printf("\nFinal Root=%f\nTotal Iterations=%d\nfunctional value=%f ",x,i+1, f(x));
         return 0;
     }
     i++;
   }while(i<=ni);
   printf("\nFinal Root=%f and functional value=%f ",x, f(x));

   return 0;
}

//Function definitions begin here
float f(float x){
  return exp(x)- x*x +3*x -2;
}

float evaluate(float x1,float x2){
   return (x2  - (f(x2)*(x2-x1)) / (f(x2) - f(x1)));
}
```

Output:

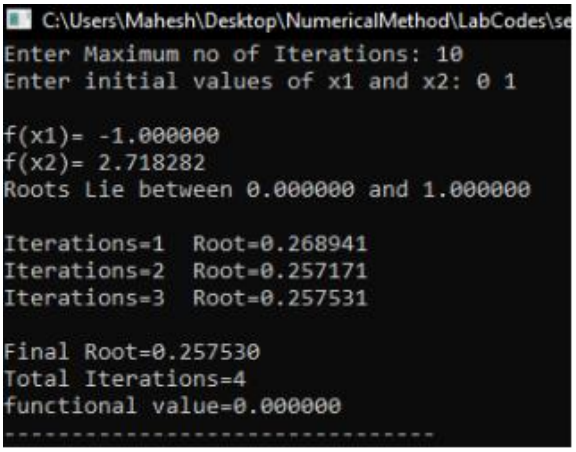
```
C:\Users\Mahesh\Desktop\NumericalMethod\LabCodes\se
Enter Maximum no of Iterations: 10
Enter initial values of x1 and x2: 0 1

f(x1)= -1.000000
f(x2)= 2.718282
Roots Lie between 0.000000 and 1.000000

Iterations=1  Root=0.268941
Iterations=2  Root=0.257171
Iterations=3  Root=0.257531

Final Root=0.257530
Total Iterations=4
functional value=0.000000
--------------------------------
```

//Example : $f(x) = x^2 - 5x - 10$

Only need to change:

```
//Function definitions begin here
float f(float x){
  return x*x - 5*x - 10;
}
```

Output:



Newton Rapson Method

Example: $f(x) = x^3 - 35$

```
#include<stdio.h>
#include<math.h>
#define e  0.00001f

float f(float x);
//This is the prototype for first derivative of f(x)
float f1(float x);


int main(){
   int in,i;
   float x,x0;
   printf("Enter Maximum no of Iterations ");
   scanf("%d",&in);

   printf("Enter the initial guess ");
      scanf("%f",&x0);

printf("\nIteration\tXn\tf(Xn)\tf1(Xn)\tXn+1\terorr
\n");
   float temp=0;

   for(i=0;i<=in;i++){
         x = x0 - (f(x0)/f1(x0));

         printf("    %d\t\t%.3f\t%.3f\t%.3f\t%.3f\t%.
3f\n",i, x0,f(x0),f1(x0),x,x-x0);

           if(fabs(x-x0)<e){
      printf("\nIterations=%d\nFinal
Root=%f\n",i,x);
       return 0;
     }

    x0=x;
   }
```

```
    return 0;
}

float f(float x){
  return x*x*x -35;
}

float f1(float x){
  return 3*x*x;
}
```

Output:

```
C:\Users\Mahesh\Desktop\NumericalMethod\LabCodes\NewtonRapson.exe
Enter Maximum no of Iterations 10
Enter the initial guess 4

Iteration       Xn      f(Xn)   f1(Xn)  Xn+1    erorr
    0           4.000   29.000  48.000  3.396   -0.604
    1           3.396   4.160   34.595  3.276   -0.120
    2           3.276   0.146   32.189  3.271   -0.005
    3           3.271   0.000   32.100  3.271   -0.000

Iterations=3
Final Root=3.271066

--------------------------------
```

Example: f(x) = xsin(x)+cos(x)

Only need to change:

```
float f(float x){
    return x*sin(x)+cos(x);
}
float f1(float x){
   return x*cos(x)+sin(x) - sin(x);
}
```

Output:

```
C:\Users\Mahesh\Desktop\NumericalMethod\LabCodes\NewtonRapson.exe
Enter Maximum no of Iterations 10
Enter the initial guess 4

Iteration       Xn      f(Xn)   f1(Xn)  Xn+1    erorr
    0           4.000   -3.681  -2.615  2.592   -1.408
    1           2.592   0.501   -2.211  2.819   0.227
    2           2.819   -0.054  -2.673  2.799   -0.020
    3           2.799   -0.000  -2.635  2.798   -0.000
    4           2.798   -0.000  -2.635  2.798   0.000

Iterations=4
Final Root=2.798386
```

CONCLUSION :

The Bisection, Secant, and Newton-Raphson methods each offer distinct advantages in root-finding. The Bisection Method guarantees convergence but can be slower. The Secant Method is faster, relying on approximated derivatives, but requires good initial guesses.

The Newton-Raphson Method converges quickly with a good starting point but depends on the derivative and may fail near inflection points. Each method has strengths and limitations, making it crucial to choose the right approach based on the specific problem.