



**Department of Computer Science and Engineering**  
Premier University

CSE 302 : Computational Methods for Engineering Problems Laboratory

Title: Implementation of Bisection, Falsi, and Newton-Raphson Methods  
for Root Finding

**Submitted by:**

<b>Name</b>	Mohammad Hafizur Rahman Sakib
<b>ID</b>	0222210005101118
<b>chapter</b>	C
<b>Session</b>	Spring 2024
<b>Semester</b>	5th Semester
<b>Submission Date</b>	14.09.2024

**Submitted to:**

Kollol Dey  
Lecturer, Department of EEE  
Premier University  
Chittagong

**Remarks**

## Introduction

This report presents the implementation and output of three widely used root-finding methods: the Bisection method, the False Position (Falsi) method, and the Newton-Raphson method. Each method is applied to solve a given mathematical equation, and the code along with its output is provided. The goal is to demonstrate how each algorithm approaches root finding without comparing their efficiencies or convergence rates.

## Bisection Method

The Bisection method is an iterative technique for finding the root of a function by repeatedly dividing an interval in half and selecting the subinterval where the root lies.

### Formula

Given a function  $f(x)$  and an interval  $[a, b]$  where  $f(a)$  and  $f(b)$  have opposite signs, the next approximation  $x_{n+1}$  is calculated using:

$$x_{n+1} = \frac{a + b}{2} \quad (0.1)$$

where: -  $a$  and  $b$  are the current interval bounds.

### Steps

1. Choose an initial interval  $[a, b]$  such that  $f(a)$  and  $f(b)$  have opposite signs.
2. Compute the midpoint  $x_{n+1}$  using:

$$x_{n+1} = \frac{a + b}{2} \quad (0.2)$$

3. Determine the subinterval  $[a, x_{n+1}]$  or  $[x_{n+1}, b]$  where the function changes sign.
4. Repeat until the interval size  $|b - a|$  is smaller than a predefined tolerance.

### Advantages

- Guarantees convergence if the initial interval contains a root.
- Simple and easy to implement.

### Limitations

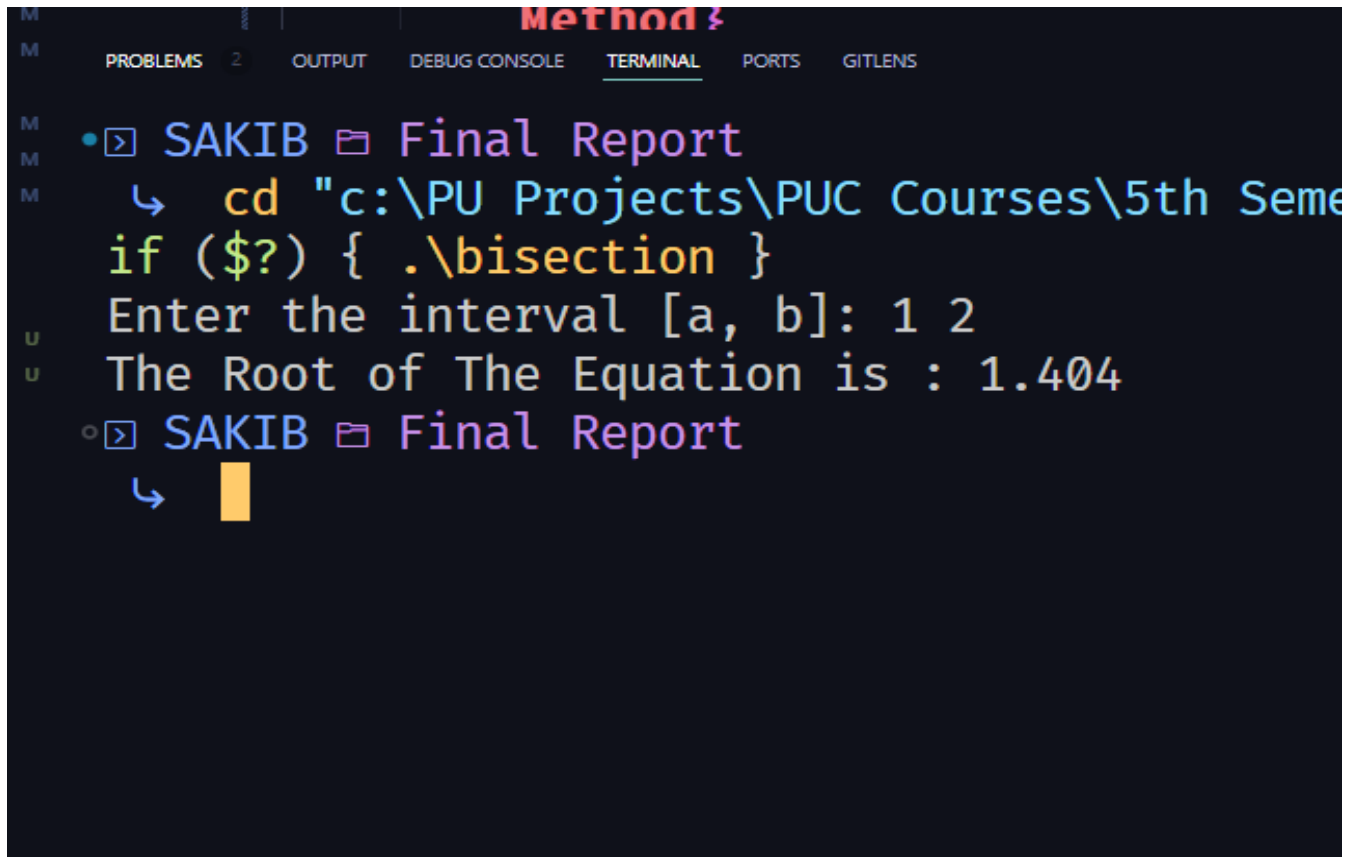
- Can be slow to converge compared to other methods.
- Requires the function to be continuous.

## Source Code :

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define setp(n) fixed << setprecision(n)
5 const double error = 1e-4;
6
7 double f(double x)
8 {
9     return ((sin(x) * sin(x)) - (x * x) + 1);
10 }
11
12 int main()
13 {
14     double x = 2.0, y = 3.0, mid = 0.00;
15     while (1)
16     {
17         cout << "Enter the interval [a, b]: ";
18         cin >> x >> y;
19         if (f(x) * f(y) < 0)
20         {
21             break;
22         }
23     }
24     while (abs(x - y) > error)
25     {
26         mid = (x + y) / 2.00;
27         if (f(mid) == 0)
28         {
29             break;
30         }
31         else if (f(mid) * f(x) < 0)
32         {
33             y = mid;
34         }
35         else
36         {
37             x = mid;
38         }
39     }
40     cout << "The Root of The Equation is : " << setp(3) << mid << endl;
41     return 0;
42 }
```

Figure : Source Code for Bisection Method

## Output :



```
Method?  
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS  
•> SAKIB Final Report  
  ↳ cd "c:\PU Projects\PUC Courses\5th Semester  
    if ($?) { .\bisection }  
    Enter the interval [a, b]: 1 2  
    The Root of The Equation is : 1.404  
◦> SAKIB Final Report  
  ↳
```

Figure : Output for Bisection Method

## False Position (Falsi) Method

The False Position method is an iterative technique for finding the roots of a function. It improves on the Bisection method by using a secant line to estimate the root.

### Formula

Given a function  $f(x)$ , the next approximation  $x_{n+1}$  is calculated using:

$$x_{n+1} = x_n - \frac{f(x_n) \cdot (x_u - x_n)}{f(x_u) - f(x_n)} \quad (0.3)$$

where: -  $x_n$  is the current approximation. -  $x_u$  is the upper bound of the interval.

### Steps

1. Choose initial guesses  $x_l$  and  $x_u$  such that  $f(x_l)$  and  $f(x_u)$  have opposite signs.
2. Compute the next approximation using:

$$x_{n+1} = x_n - \frac{f(x_n) \cdot (x_u - x_n)}{f(x_u) - f(x_n)} \quad (0.4)$$

3. Update the interval based on the sign of  $f(x_{n+1})$ .
4. Repeat until the difference  $|x_{n+1} - x_n|$  is smaller than a predefined tolerance.

## Advantages

- Faster convergence than the Bisection method.
- Does not require derivative information.

## Limitations

- May converge slowly if the function is not well-behaved.
- Requires an initial interval with a sign change.

## Source Code :

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 const double error = 1e-5;
5 double f(double x)
6 {
7     return ((sin(x) * sin(x)) - (x * x) + 1);
8 }
9
10 int main()
11 {
12     double Xo, X1, X;
13     cout << "Enter the interval [a, b]: ";
14     cin >> Xo >> X1;
15     if ((f(Xo) * f(X1)) < 0)
16     {
17         while (abs(f(X)) > error)
18         {
19             X = (((Xo * f(X1)) - (X1 * f(Xo))) / (f(X1) - f(Xo)));
20             if ((f(X1) * f(X)) < 0)
21             {
22                 Xo = X;
23             }
24             else
25             {
26                 X1 = X;
27             }
28         }
29         cout << "Root = " << X << endl;
30     }
31     else
32     {
33         cout << "Enter The Initial Values Again : " << endl;
34         cin >> Xo >> X1;
35     }
36     return 0;
37 }
```

Figure : Source Code for Falsi Method

## Output :

```
> SAKIB Final Report
• ↳ cd "c:\PU Projects\PUC Courses\5t
if ($?) { .\falsi }
Enter the interval [a, b]: 1 2
Root = 1.40449
◦> SAKIB Final Report
↳
```

Figure : Output for Falsi Method

## Newton-Raphson Method

The Newton-Raphson method is an iterative technique for approximating the roots of a function. It uses the function's derivative to refine guesses for the root.

### Formula

Given a function  $f(x)$  and its derivative  $f'(x)$ , the iteration formula is:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (0.5)$$

### Steps

1. Start with an initial guess  $x_0$ .
2. Update the guess using:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (0.6)$$

3. Repeat until  $|x_{n+1} - x_n|$  is smaller than a predefined tolerance.

### Advantages

- Fast convergence near the root.
- Simple to implement.

### Limitations

- Requires a good initial guess.
- Needs the derivative of the function.

## Source Code :

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 double f(double x)
5 {
6     return (sin(x) * sin(x) - x * x + 1);
7 }
8
9 double g(double x)
10 {
11     return (2 * sin(x) * cos(x)) - (2 * x);
12 }
13
14 int main()
15 {
16     double guess1, guess2, guess, ans = 0.00;
17
18     cout << "Enter the interval [a, b]: ";
19     cin >> guess1 >> guess2;
20     if (f(guess1) * f(guess2) > 0)
21     {
22         cout << "No root exists between " << guess1 << " and " << guess2 << endl;
23         return 0;
24     }
25
26     guess = guess1;
27     while (1)
28     {
29         double fg = f(guess);
30         double gg = g(guess);
31         double x1 = guess - (fg / gg);
32         if ((abs((x1 - guess) / x1)) <= 1e-8)
33         {
34             ans = x1;
35             break;
36         }
37         else
38         {
39             guess = x1;
40         }
41     }
42
43     cout << fixed << setprecision(8);
44     cout << "Root = " << ans << endl;
45
46     return 0;
47 }
48
```

Figure : Source Code for Newton Raphson Method

## Output :

```
PROBLEMS 20 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
SAKIB Final Report
• ↳ cd "c:\PU Projects\PUC Courses\5th Semester\CMEPL\Final Report\
ntial } ; if ($?) { .\tengential }
Enter the interval [a, b]: 1 2
Root = 1.40449165
SAKIB Final Report
↳
```

Figure : Output for Newton Raphson Method

## Conclusion

This report demonstrates the implementation of the Bisection, False Position, and Newton-Raphson methods for solving a root-finding problem. Each method successfully finds the root, and the code and outputs are included to show the process and results. Further analysis could compare the efficiency and convergence behavior of these methods.