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:

Name	Mohammad Hafizur Rahman Sakib
ID	0222210005101118
Section	C
Session	Spring 2024
Semester	5th Semester
Submission Date	15.09.2024

Submitted to:
Kollol Dey
Assistant Professor, Department of Mathematics
Premier University
Chittagong

Remarks

Objective

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:

$$x_{n+1} = \frac{a + b}{2} \tag{0.1}$$

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:

$$x_{n+1} = \frac{a + b}{2} \tag{0.2}$$

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.

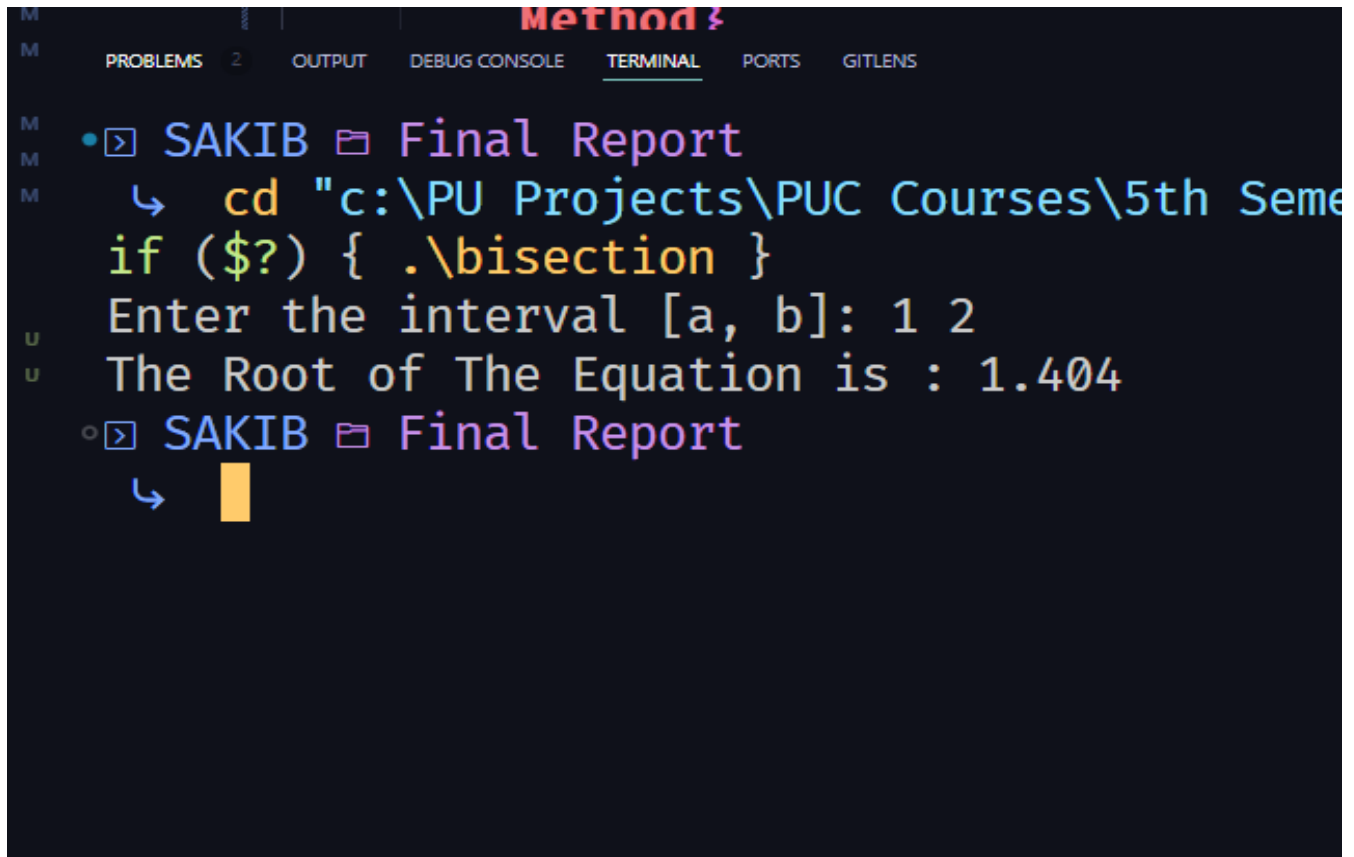- Simple and easy to implement.

Limitations

- Can be slow to converge compared to other methods.

- Requires the function to be continuous.

Source Code :

```cpp
#include <bits/stdc++.h>
using namespace std;

#define setp(n) fixed << setprecision(n)
const double error = 1e-4;

double f(double x)
{
    return ((sin(x) * sin(x)) - (x * x) + 1);
}

int main()
{
    double x = 2.0, y = 3.0, mid = 0.00;
    while (1)
    {
        cout << "Enter the interval [a, b]: ";
        cin >> x >> y;
        if (f(x) * f(y) < 0)
        {
            break;
        }
    }
    while (abs(x - y) > error)
    {
        mid = (x + y) / 2.00;
        if (f(mid) == 0)
        {
            break;
        }
        else if (f(mid) * f(x) < 0)
        {
            y = mid;
        }
        else
        {
            x = mid;
        }
    }
    cout << "The Root of The Equation is : " << setp(3) << mid << endl;
    return 0;
}
```

Figure : Source Code for Bisection Method

Output :

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:

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

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:

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

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.

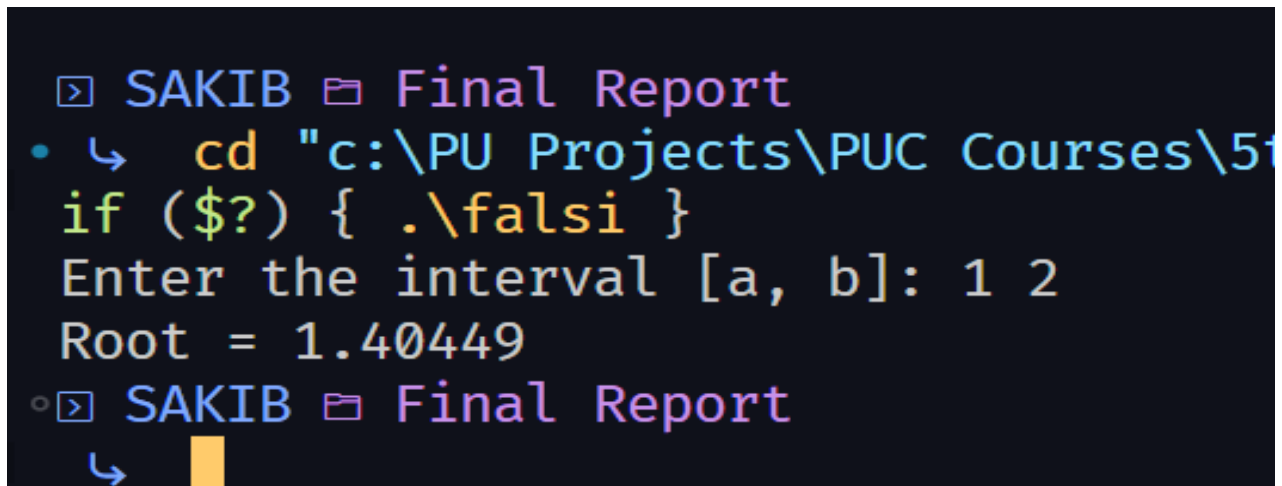- Does not require derivative information.

Limitations

- May converge slowly if the function is not well-behaved.

- Requires an initial interval with a sign change.

Source Code :

```cpp
#include <bits/stdc++.h>
using namespace std;

const double error = 1e-5;
double f(double x)
{
    return ((sin(x) * sin(x)) - (x * x) + 1);
}

int main()
{
    double Xo, X1, X;
    cout << "Enter the interval [a, b]: ";
    cin >> Xo >> X1;
    if ((f(Xo) * f(X1)) < 0)
    {
        while (abs(f(X)) > error)
        {
            X = (((Xo * f(X1)) - (X1 * f(Xo))) / (f(X1) - f(Xo)));
            if ((f(X1) * f(X)) < 0)
            {
                Xo = X;
            }
            else
            {
                X1 = X;
            }
        }
        cout << "Root = " << X << endl;
    }
    else
    {
        cout << "Enter The Initial Values Again : " << endl;
        cin >> Xo >> X1;
    }
    return 0;
}
```

Figure : Source Code for Falsi Method

5

Output :

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:

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

Steps

1. Start with an initial guess x_0.

2. Update the guess using:

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

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 :

```cpp
#include <bits/stdc++.h>
using namespace std;

double f(double x)
{
    return (sin(x) * sin(x) - x * x + 1);
}

double g(double x)
{
    return (2 * sin(x) * cos(x)) - (2 * x);
}

int main()
{
    double guess1, guess2, guess, ans = 0.00;

    cout << "Enter the interval [a, b]: ";
    cin >> guess1 >> guess2;
    if (f(guess1) * f(guess2) > 0)
    {
        cout << "No root exists between " << guess1 << " and " << guess2 << endl;
        return 0;
    }

    guess = guess1;
    while (1)
    {
        double fg = f(guess);
        double gg = g(guess);
        double x1 = guess - (fg / gg);
        if ((abs((x1 - guess) / x1)) <= 1e-8)
        {
            ans = x1;
            break;
        }
        else
        {
            guess = x1;
        }
    }

    cout << fixed << setprecision(8);
    cout << "Root = " << ans << endl;

    return 0;
}
```

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

Discussion

In this report, we have demonstrated the implementation of three root-finding methods: the Bisection method, the False Position (Falsi) method, and the Newton-Raphson method. Each method offers a unique approach to approximating the roots of a function:

- The Bisection method provides a reliable but slower approach by iteratively narrowing down an interval where the root lies.

- The False Position method improves upon the Bisection method by using a secant line to enhance convergence speed while still requiring an interval with a sign change.

- The Newton-Raphson method offers rapid convergence near the root by using the function's derivative, though it depends on a good initial guess and the availability of the derivative.

The provided code and outputs illustrate the practical application of these methods, showcasing their ability to find roots effectively. While the Bisection and False Position methods are useful for their simplicity and reliability, the Newton-Raphson method stands out for its efficiency in converging to the root under suitable conditions.

Overall, this report highlights the versatility of these root-finding techniques and their applicability in solving mathematical equations. Each method has its advantages and limitations, making them suitable for different scenarios based on the problem's requirements and characteristics.