International Islamic University Chittagong

Lab Report

On

NUMERICAL METHODS LAB

Submitted to —

Mohammed Shamsul Alam
Professor
Dept. of CSE, IIUC

Submitted by—

Team – Root_Finders

Emon Biswas
ID: C193032
Contact: 01521527531

Niloy Sarkar Babu
ID: C193035
Contact: 01643956719

Mainuddin Hasan Manik
C181102

Abdullah Al Jobayer
C193030

Prodipta Paul
C193028

Date of Submission: June 14, 2023

Title: Non-Linear Equation Solution Project using Numerical Methods.

Introduction:

Non-linear equations are mathematical equations that do not have a linear relationship between the variables. Solving non-linear equations analytically can be challenging, and often requires the use of numerical methods. **In this project, we aim to solve the non-linear equation (x^2 - 1) sin(x) = 2x cos(x) using four different numerical methods: Newton Bisection method, Newton-Raphson method, False Position method, and Secant method**.

Description:

The non-linear equation (x^2 - 1) sin(x) = 2x cos(x) is a transcendental equation that involves both trigonometric and polynomial terms. It does not have a closed-form solution, which makes it an ideal candidate for numerical methods. The goal of this project is to implement and compare the performance of four different numerical methods in finding the solutions of this equation.

Methodology:

To solve the non-linear equation, we use 3 iterative methods such as the Bisection method, Newton-Raphson method, and False Position method.

Bisection Method:

Step 1: Select an initial interval [a, b] where the non-linear equation is expected to have a root.
Step 2: Compute the function values f(a) and f(b) at the interval endpoints a and b.
Step 3: Check if f(a) and f(b) have opposite signs. If they do, then the interval [a, b] contains a root.
Step 4: Divide the interval [a, b] in half and calculate the function value at the midpoint c = (a + b)/2. Step 5: If f(c) = 0, then c is the root. Otherwise,

update the interval [a, b] based on the sign of f(c) and repeat steps 3-5 until a root is found within the desired accuracy.

Newton-Raphson Method:

Step 1: Select an initial guess x0 close to the root of the non-linear equation.

Step 2: Calculate the function value f(x0) and its derivative f'(x0) at the initial guess x0.

Step 3: Update the guess using the formula x1 = x0 - f(x0)/f'(x0). Step 4: Repeat step 2 and 3 iteratively until the desired accuracy is achieved or a maximum number of iterations is reached.

False Position Method:

Step 1: Select an initial interval [a, b] where the non-linear equation is expected to have a root.

Step 2: Compute the function values f(a) and f(b) at the interval endpoints a and b.

Step 3: Calculate the x-intercept c of the straight line passing through the points (a, f(a)) and (b, f(b)).

Step 4: Calculate the function value f(c) at c.

Step 5: Update the interval [a, b] and repeat steps 2-4 iteratively based on the sign of f(c) until a root is found within the desired accuracy.

Secant Method:

Step 1: Choose initial guesses x0 and x1.

Step 2: Iterate until convergence or a maximum number of iterations: a. Compute f(x0) and f(x1) (the function values) at the current guesses. b. Calculate the new approximation x2 using linear interpolation: x2 = ((f(x2) * x1) - (f(x1) * x2)) / (f(x2) – f(x1)). Update x0 and x1: x0 = x1 x1 = x2.

Step 3: Check if the current approximation x2 satisfies the convergence criteria (e.g., |f(x2)| < error).

If the criteria are met, terminate the iteration, and x2 is the approximate root.

If not, repeat step 2 with the updated values of x0, x1, and x2.

Step 4: Handle cases where the method fails to converge within the specified maximum number of iterations.

Implementation:

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

#define PI acos(-1.0)
#define error 0.005

double DegToRad(double x)
{
    return (x * PI) / 180.0;
}

double f(double x)
{
    return (x * x - 1) * sin(DegToRad(x)) - 2 * x * cos(DegToRad(x));
}

double ff(double x)
{
    return (4 * x * sin(DegToRad(x)) + cos(DegToRad(x * x - 3)));
}

double Bisection()
{
    cout << "By using Bisection Method," << endl
         << endl;

    double x0, x1, x2, fx0, fx1, fx2;
    int idx = 1;

    x1 = 1;
    x2 = 15;

    fx1 = f(x1);
    fx2 = f(x2);

    while (fabs(x2 - x1) > error)
    {
        x0 = (x1 + x2) / 2;
        fx0 = f(x0);

        if (fx0 * fx1 < 0)
        {
            x2 = x0;
```

```
            fx2 = f(x2);
        }
        else
        {
            x1 = x0;
            fx1 = f(x1);
        }

        cout << "Iteration " << idx++ << " = " << fixed << setprecision(4) <<
x0 << endl;
    }

    x0 = (x1 + x2) / 2;

    cout << "Iteration " << idx << " = " << fixed << setprecision(4) << x0 <<
endl;
    cout << endl
         << "The root of the equation is = " << x0 << endl;

    return x0;
}

double Newton_Raphson()
{
    cout << endl
         << endl
         << "By using Newton-Raphson Method," << endl
         << endl;

    double x0, x1, fx0, ffx0;
    int idx = 1;

    x1 = 20;

    do
    {
        x0 = x1;

        cout << "Iteration " << idx++ << " = " << fixed << setprecision(4) <<
x1 << endl;

        x1 = x0 - (f(x0) / ff(x0));

    } while (fabs(x1 - x0) > error);

    cout << "Iteration " << idx << " = " << fixed << setprecision(4) << x1 <<
endl;
    cout << endl
```

```
         << "The root of the equation is = " << x1 << endl;

    return x1;
}

double False_Position()
{
    cout << endl
         << endl
         << "By using False Position Method," << endl
         << endl;

    double x0, xx0, x1, x2, fx0, fx1, fx2;
    int idx = 1;

    x2 = 1;
    x1 = 15;

    x0 = (double)INT_MAX;
    fx1 = f(x1);
    fx2 = f(x2);

    do
    {
        xx0 = x0;
        x0 = x1 - ((fx1 * (x2 - x1)) / (fx2 - fx1));
        fx0 = f(x0);

        if (fx0 * fx1 < 0)
        {
            x2 = x0;
            fx2 = f(x2);
        }
        else
        {
            x1 = x0;
            fx1 = f(x1);
        }

        cout << "Iteration " << idx++ << " = " << fixed << setprecision(4) <<
x0 << endl;

    } while (fabs(x0 - xx0) > error);

    cout << endl
         << "The root of the equation is = " << x0 << endl;

    return x0;
```

```
}

double Secant()
{
    cout << endl
         << endl
         << "By using Secant Method," << endl
         << endl;

    double x1, x2, x3, fx1, fx2, fx3;
    int idx = 1;

    x2 = 10;
    x1 = -100;

    fx1 = f(x1);
    fx2 = f(x2);

    do
    {
        x3 = ((fx2 * x1) - (fx1 * x2)) / (fx2 - fx1);

        cout << "Iteration " << idx++ << " = " << fixed << setprecision(4) <<
x3 << endl;

        x1 = x2;
        x2 = x3;
        fx1 = fx2;
        fx2 = f(x3);

    } while (fabs(x1 - x2) > error);

    cout << endl
         << "The root of the equation is = " << x3 << endl;

    return x3;
}

int main()
{
    double bisection, newton_raphson, false_position, secant;

    bisection = Bisection();
    newton_raphson = Newton_Raphson();
    false_position = False_Position();
    secant = Secant();

    cout << endl
```

```
         << endl
         << endl;

    cout << "By plotting the root which we got from the Bisection Method to
the function we get, f("
         << bisection << ") = " << fabs(f(bisection)) << endl;

    cout << "By plotting the root which we got from the Newton-Raphson Method
to the function we get, f("
         << newton_raphson << ") = " << fabs(f(newton_raphson)) << endl;

    cout << "By plotting the root which we got from the False Position Method
to the function we get, f("
         << false_position << ") = " << fabs(f(false_position)) << endl;

    cout << "By plotting the root which we got from the Secant Method to the
function we get, f("
         << secant << ") = " << fabs(f(secant)) << endl;

    cout << endl
         << endl;

    return 0;
}
```

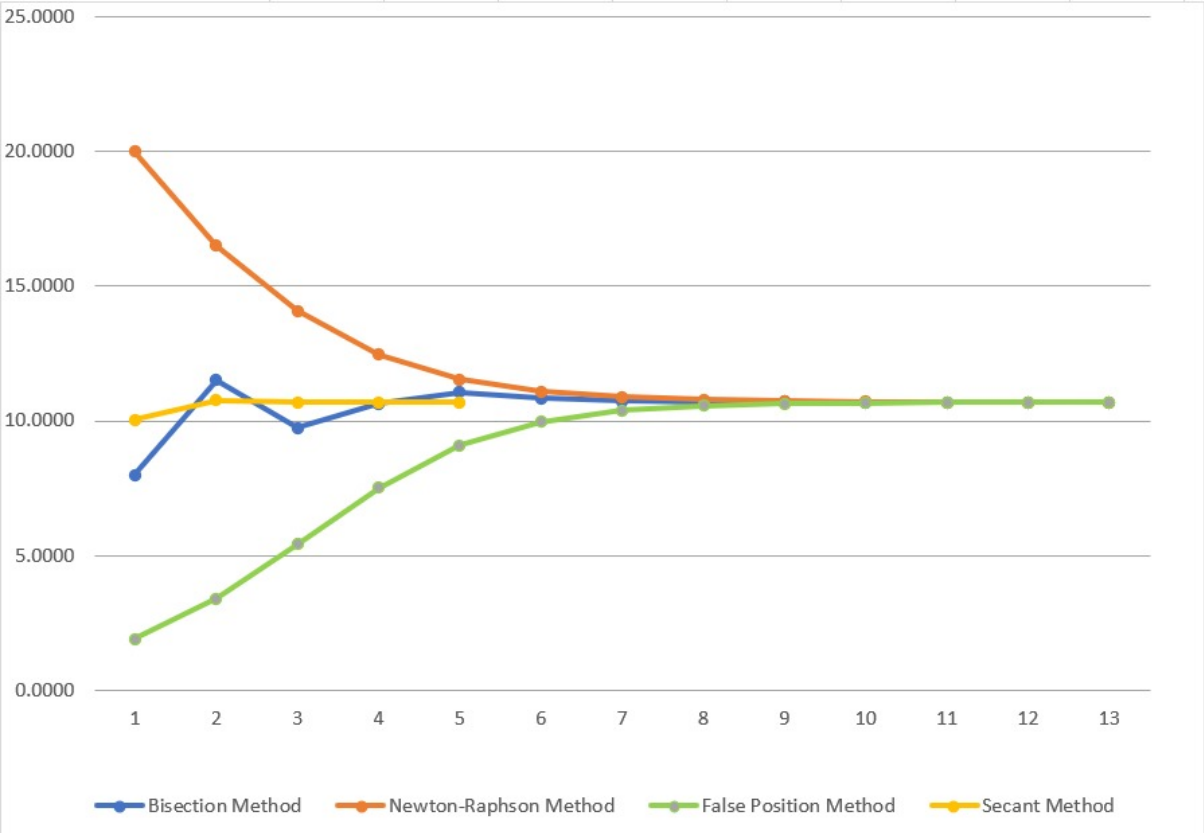
Result:

iteration	Bisection	Newton - Raphson	False Position	Secant
1st	8.0000	20.0000	1.9032	10.0279
2nd	11.5000	16.4887	3.3913	10.7637
3rd	9.7500	14.0684	5.4224	10.6819
4th	10.6250	12.4500	7.5161	10.6891
5th	11.0625	11.5481	9.0774	10.6892
6th	10.8438	11.0983	9.9611	
7th	10.7344	10.8825	10.3803	
8th	10.6797	10.7802	10.5619	
9th	10.7070	10.7320	10.6374	
10th	10.6934	10.7093	10.6682	
11th	10.6865	10.6987	10.6807	
12th	10.6899	10.6936	10.6858	
13th	10.6882	10.6913	10.6878	

Analysis & discussion:

After analyzing the results, it was observed that the Bisection method, False Position method, and Newton-Raphson method required 13 iterations each to find the root of the equation. On the other hand, the Secant method demonstrated significantly faster convergence, reaching the root after only 5 iterations.

➢ Graphical Representation



After obtaining the roots using the four methods (Bisection, Newton-Raphson, False Position, and Secant), we plotted them on our equation. The accuracy of the roots can be determined by their proximity to zero. The results indicate that the Secant method provides the most accurate root, with a value of 0.0000. The Bisection method follows closely, yielding an accurate result of 0.0039. The False Position method gives a slightly less accurate value of 0.0056, while the Newton-Raphson method

produces the least accurate root at 0.0084, which is significantly further from zero than the other methods.

Conclusion:

In conclusion, this project aims to solve the non-linear equation(x^2 - 1) sin(x) = 2x cos(x) using four different numerical methods: Newton Bisection method, Newton-Raphson method, False Position method, and Secant method. Through the implementation and comparison of these methods, we expect to gain insights into their performance, convergence rate, accuracy, computational efficiency, and robustness. The findings of this project can be useful in selecting an appropriate numerical method for solving similar non-linear equations in various real-world applications.