Non-Linear Equation Solution Project using Numerical Methods.

Submitted To

Mohammed Shamsul Alam
Professor
Dept. of CSE, IIUC

Submitted By

ROOT_FINDERS

Emon Biswas (C193032)
Leader

Niloy Sarkar Babu (C193035)
Deputy Leader

Mainuddin Hasan Manik (C181102)

Abdullah Al Jobayer (C193030)

Prodipta Paul (C193028)

ABSTRACT

This project focuses on finding the root of the equation **$(x^2 - 1) \sin(x) = 2x\cos(x)$** using four different numerical methods: **Bisection, Newton-Raphson, False Position, and Secant**. The results obtained from each method are compared and analyzed to determine the most effective approach.

The study begins by applying the four methods to the equation with various initial guesses. The outcomes, number of iterations, and approximate root value, are summarized in a table. Additionally, a graphical representation of the equation and the convergence of each method towards the root is provided.

Upon analyzing the results, it is evident that all four methods successfully find a root for the equation. However, there are differences in terms of convergence speed and accuracy. The Secant method stands out as the most efficient and accurate, followed by the Bisection method, False Position method, and Newton-Raphson method.

INTRODUCTION

A non-linear equation is a mathematical equation that does not follow the basic form of a linear equation, where the unknown variable(s) appears in a power other than 1. Non-linear equations can have multiple solutions, which may be real or complex. Non-linear equations can be solved analytically using various mathematical techniques, but in many cases, numerical methods such as the Bisection method, Newton-Raphson method, False Position method and Secant method are used to approximate the solution. Non-linear equations arise n various fields of science and engineering, and their solution is essential for modeling and understanding many natural phenomena.

METHODOLOGY

To solve the non-linear equation, we use 4 iterative methods such as the Bisection method, Newton-Raphson method, and False Position method, Secant 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)) by using the formula **c = a - ((f(a) * (b - a)) / (f(b) – f(a))**.
- 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)));
}
```

```
22  double Bisection()
23  {
24      cout << "By using Bisection Method," << endl
25           << endl;
26
27      double x0, x1, x2, fx0, fx1, fx2;
28      int idx = 1;
29
30      x1 = 1;
31      x2 = 15;
32
33      fx1 = f(x1);
34      fx2 = f(x2);
35
36      while (fabs(x2 - x1) > error)
37      {
38          x0 = (x1 + x2) / 2;
39          fx0 = f(x0);
40
41          if (fx0 * fx1 < 0)
42          {
43              x2 = x0;
44              fx2 = f(x2);
45          }
46          else
47          {
48              x1 = x0;
49              fx1 = f(x1);
50          }
51
52          cout << "Iteration " << idx++ << " = " << fixed << setprecision(4) << x0 << endl;
53      }
54
55      x0 = (x1 + x2) / 2;
56
57      cout << "Iteration " << idx << " = " << fixed << setprecision(4) << x0 << endl;
58      cout << endl
59           << "The root of the equation is = " << x0 << endl;
60
61      return x0;
62  }
```

```
64  double Newton_Raphson()
65  {
66      cout << endl
67           << endl
68           << "By using Newton-Raphson Method," << endl
69           << endl;
70
71      double x0, x1, fx0, ffx0;
72      int idx = 1;
73
74      x1 = 20;
75
76      do
77      {
78          x0 = x1;
79
80          cout << "Iteration " << idx++ << " = " << fixed << setprecision(4) << x1 << endl;
81
82          x1 = x0 - (f(x0) / ff(x0));
83
84      } while (fabs(x1 - x0) > error);
85
86      cout << "Iteration " << idx << " = " << fixed << setprecision(4) << x1 << endl;
87      cout << endl
88           << "The root of the equation is = " << x1 << endl;
89
90      return x1;
91  }
```

```cpp
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;
}
```

```cpp
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;
}
```

```cpp

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;
}
```

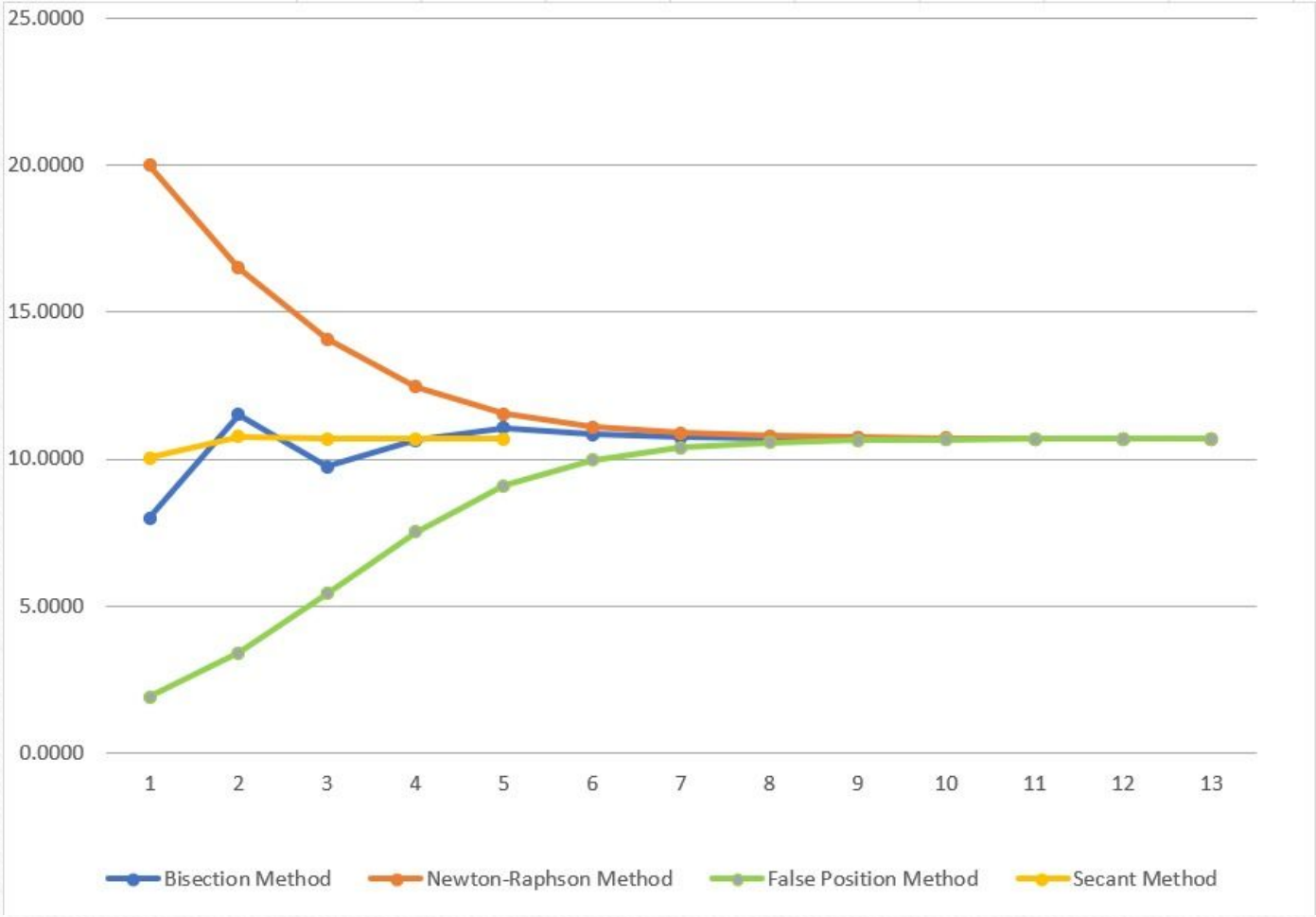
RESULT

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.

Snapshot of the output

```
By plotting the root which we got from the Bisection Method to the function we get, f(10.6882) = 0.0039
By plotting the root which we got from the Newton-Raphson Method to the function we get, f(10.6913) = 0.0084
By plotting the root which we got from the False Position Method to the function we get, f(10.6878) = 0.0056
By plotting the root which we got from the Secant Method to the function we get, f(10.6892) = 0.0000
```

CONCLUSION

In conclusion, our project focused on finding the root of the equation $(x^2 - 1) \sin(x) = 2x\cos(x)$ using four numerical methods: **Bisection, Newton-Raphson, False Position, and Secant**. After analyzing the results, it was found that the Secant method provided the most accurate and efficient solution, converging to the root in just 5 iterations. The Bisection method followed closely, while the False Position method was slightly less accurate. The Newton-Raphson method demonstrated the lowest accuracy among the four methods.

The findings suggest that the choice of numerical method is crucial in obtaining accurate roots. The Secant method is recommended for this specific equation due to its rapid convergence and high precision. However, the Bisection and False Position methods can also be suitable options, depending on the desired trade-off between speed and accuracy. The Newton-Raphson method, while less accurate in this case, may have advantages for different types of equations.

THANK YOU!