Lab 01 - Bisection and False Position Methods for the Solution of Non-Linear Equations

Objective

To understand and implement the Bisection and False Position methods for finding roots of non-linear equations:

1. Implement the **Bisection method** to approximate a root of a continuous function.
2. Implement the **False Position method** to approximate a root of the same function.
3. Compare the convergence behaviour and accuracy of both methods.

Theory

Bisection Method

Bisection method is the half interval method based on **Intermediate Value Theorem** which states that if a function is continous in the interval `a` and `b` such that `f(a)` gives positive functional value and `f(b)` gives negative functional value then `f(a) * f(b) < 0` and a root lies in between `a` and `b`.

For the bisection method, the approximation to root is calculated by using midpoint formula: $x_0 = \frac{a+b}{2}$

Advantages:

- Guarantees convergence as the interval is halved in each iteration.
- Simple to implement and does not require derivative calculations.
- Robust for continuous functions with a root in the interval.

Limitations:

- Convergence can be slow, especially for functions with flat regions near the root.
- Requires the root to be bracketed initially, which may not always be straightforward.

The Bisection method is particularly effective for functions where guaranteed convergence is more important than speed.

False Position Method

The False Position method, also known as the **Regula Falsi method**, is a root-finding algorithm that combines the principles of the **Bisection method** and **linear interpolation**. It is based on the assumption that the function is approximately linear in the interval `[a, b]` where `f(a)` and `f(b)` have opposite signs, i.e., `f(a) * f(b) < 0`.

The root is approximated by the point where the straight line connecting `(a, f(a))` and `(b, f(b))` crosses the x-axis. The formula for the root is:

$x_0 = \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)}$

Advantages:

- Faster convergence compared to the Bisection method for well-behaved functions.
- Retains the bracketing property, ensuring the root lies within the interval.

Limitations:

- Convergence may slow down if the function is highly non-linear, as one endpoint may remain fixed for several iterations.
- Requires two function evaluations per iteration, increasing computational cost compared to Bisection.

The False Position method is particularly effective for functions that are approximately linear in the region of interest.

Algorithm

Bisection Method

1. Choose initial bracket where the root lies, `a` and `b` i.e. `f(a) * f(b) < 0`.
2. Midpoint formula: $x_0 = \frac{a+b}{2}$
3. Check the sign of $f(x_0)$:
   - If $f(a) * f(x0) < 0$ then bracket is (a, x_0) i.e. $b = x_0$.
   - If $f(b) * f(x_0) < 0$ then $a = x_0$.
   - If $f(x_0) == 0$, x_0 is root.
4. Repeat step 2 and step 3 until tolerance is met.

False Position Method

1. Choose initial guesses a and b such that $f(a) * f(b) < 0$.
2. Compute the root approximation using the formula above.
3. Check the sign of $f(x_0)$:
   - If $f(a) * f(x_0) < 0$, update $b = x_0$.
   - If $f(b) * f(x_0) < 0$, update $a = x_0$.
   - If $f(x_0) == 0$, x_0 is the root.
4. Repeat until the desired tolerance is achieved.

Source Code

Bisection Method

```c
#include <stdio.h>
#include <math.h>
#define TOLERANCE 0.0001

double f(double x)
{
    // function: f(x) = x^2 - x - 2
    return x * x - x - 2;

    // function: f(x) = x - e^-x
    // return x - exp(-x);

    // function: f(x) = x e^x - cos(x)
    // return x * exp(x) - cos(x);

    // function: f(x) = x log10(x) - 1.2
    // return x * log10(x) - 1.2;
}

int main(int argc, char const *argv[])
{
    double a, b, x, fa, fb, fx; // Initial guesses and function values
    int iter = 0;               // Iteration counter

    // Get the initial valid guesses from the user
    while (1)
    {
        // Input the initial guesses
        printf("\nEnter the initial guesses a and b: ");
        scanf("%lf %lf", &a, &b);

        // Get the function values at the initial guesses
        fa = f(a);
        fb = f(b);

        // Check if the initial guesses are valid
        if (fa * fb > 0)
        {
            printf("Invalid initial guesses. f(a) and f(b) must have opposite signs.\n");
        }
        else
        {
            printf("Valid initial guesses.\n");
            break;
        }
    }

    // Bisection method
    printf("\nBisection Method:\n");
    printf("Iteration\t a\t\t b\t\t x\t\t f(x)\t\t |f(x)|\n");
    printf("-------------------------------------------------------------------------------------------\n");
    do
    {
        // Calculate the midpoint
        x = (a + b) / 2.0;
        fx = f(x);

        // Print the current iteration values
        printf("%d\t\t %.6f\t %.6f\t %.6f\t %.6f\t %.6f\n", iter, a, b, x, fx, fabs(fx));
```

```
        // Check if the root is found or if the tolerance is met
        if (fabs(fx) < TOLERANCE || fabs(b - a) < TOLERANCE)
            break;

        // Update the guesses based on the function values
        if (fa * fx < 0)
        {
            b = x;
            fb = fx;
        }
        else
        {
            a = x;
            fa = fx;
        }

        iter++;

    } while (1);

    printf("-----------------------------------------------------------------------------------------\n");
    // Print the final result
    printf("Root found: x = %.6f, f(x) = %.6f\n", x, fx);

    return 0;
}
```

Output:

```
Enter the initial guesses a and b: 3 7
Invalid initial guesses. f(a) and f(b) must have opposite signs.

Enter the initial guesses a and b: 45 89
Invalid initial guesses. f(a) and f(b) must have opposite signs.

Enter the initial guesses a and b: 1 2.5
Valid initial guesses.

Bisection Method:
Iteration       a               b               x               f(x)            |f(x)|
-----------------------------------------------------------------------------------------
0               1.000000        2.500000        1.750000        -0.687500       0.687500
1               1.750000        2.500000        2.125000        0.390625        0.390625
2               1.750000        2.125000        1.937500        -0.183594       0.183594
3               1.937500        2.125000        2.031250        0.094727        0.094727
4               1.937500        2.031250        1.984375        -0.046631       0.046631
5               1.984375        2.031250        2.007813        0.023499        0.023499
6               1.984375        2.007813        1.996094        -0.011703       0.011703
7               1.996094        2.007813        2.001953        0.005863        0.005863
8               1.996094        2.001953        1.999023        -0.002929       0.002929
9               1.999023        2.001953        2.000488        0.001465        0.001465
10              1.999023        2.000488        1.999756        -0.000732       0.000732
11              1.999756        2.000488        2.000122        0.000366        0.000366
12              1.999756        2.000122        1.999939        -0.000183       0.000183
13              1.999939        2.000122        2.000031        0.000092        0.000092
-----------------------------------------------------------------------------------------
Root found: x = 2.000031, f(x) = 0.000092
```

False Position Method

```c
#include <stdio.h>
#include <math.h>
#define TOLERANCE 0.0001

double f(double x)
{
    // function: f(x) = x^2 - x - 2
    return x * x - x - 2;

    // function: f(x) = x - e^-x
    // return x - exp(-x);

    // function: f(x) = x e^x - cos(x)
    // return x * exp(x) - cos(x);

    // function: f(x) = x log10(x) - 1.2
    // return x * log10(x) - 1.2;
}

int main(int argc, char const *argv[])
{
    double a, b, x, fa, fb, fx; // Initial guesses and function values
    int iter = 0;               // Iteration counter

    // Get the initial valid guesses from the user
    while (1)
    {
        // Input the initial guesses
        printf("\nEnter the initial guesses a and b: ");
        scanf("%lf %lf", &a, &b);

        // Get the function values at the initial guesses
        fa = f(a);
        fb = f(b);

        // Check if the initial guesses are valid
        if (fa * fb > 0)
        {
            printf("Invalid initial guesses. f(a) and f(b) must have opposite signs.\n");
        }
        else
        {
            printf("Valid initial guesses.\n");
            break;
        }
    }

    // False position method
    printf("\nFalse Position Method:\n");
    printf("Iteration\t a\t\t b\t\t x\t\t f(x)\t\t |f(x)|\n");
    printf("-------------------------------------------------------------------------------------------\n");
    do
    {
        // Calculate the midpoint
        x = (a * fb - b * fa) / (fb - fa);
        fx = f(x);

        // Print the current iteration values
        printf("%d\t\t %.6f\t %.6f\t %.6f\t %.6f\t %.6f\n", iter, a, b, x, fx, fabs(fx));

        // Check if the root is found or if the tolerance is met
        if (fabs(fx) < TOLERANCE || fabs(b - a) < TOLERANCE)
            break;
```

```
        // Update the guesses based on the function values
        if (fa * fx < 0)
        {
            b = x;
            fb = fx;
        }
        else
        {
            a = x;
            fa = fx;
        }

        iter++;

    } while (1);

    printf("-----------------------------------------------------------------------------------------\n");
    // Print the final result
    printf("Root found: x = %.6f, f(x) = %.6f\n", x, fx);

    return 0;
}
```

Output:

```
Enter the initial guesses a and b: 1 2.5
Valid initial guesses.

False Position Method:
Iteration       a               b               x               f(x)            |f(x)|
-----------------------------------------------------------------------------------------
0               1.000000        2.500000        1.800000        -0.560000       0.560000
1               1.800000        2.500000        1.969697        -0.089991       0.089991
2               1.969697        2.500000        1.995633        -0.013081       0.013081
3               1.995633        2.500000        1.999375        -0.001873       0.001873
4               1.999375        2.500000        1.999911        -0.000268       0.000268
5               1.999911        2.500000        1.999987        -0.000038       0.000038
-----------------------------------------------------------------------------------------
Root found: x = 1.999987, f(x) = -0.000038
```

Discussion

- **Convergence Speed:** False Position required only 5 iterations versus 13 for Bisection, demonstrating faster convergence on this test function, $f(x) = x^2 - x - 2$
- **Reliability:** Bisection consistently reduces the bracketing interval by half, ensuring steady progress regardless of function shape. But, False Position may update only one endpoint repeatedly when one side's function value is far larger in magnitude, slowing interval reduction.
- **Accuracy:** Both methods achieved similar accuracy of tolerance `|f(x)| < 0.0001` .
- **Computational Cost:** Each iteration requires one function evaluation for Bisection and two for False Position (at both endpoints), so trade-offs exist between iterations and evaluations as the Bisection method has more iterations than False Position method.

Conclusion

- The Bisection method is robust and guarantees convergence but may require more iterations.
- The False Position method can converge faster for well-behaved functions but is not guaranteed to reduce the interval size symmetrically.

- For the test functions , False Position outperformed Bisection in speed while delivering comparable accuracy.
- In practice, algorithm choice may depend on the trade-off between robustness and speed, as well as function characteristics.

Submitted By: Kushal Prasad Joshi
Roll No: 230345
Faculty: Science and Technology
Program: BE Computer
Group: B