Numerical Root-Finding Methods in Java: Bisection and False-Position

Mikolaj Suchon

23/03/2025

1 Introduction

Root-finding methods are fundamental numerical techniques used in scientific computing to approximate solutions to equations of the form $f(x) = 0$. This report details the implementation of two such methods in Java: the **Bisection Method** and the **False-Position Method**. These methods estimate roots iteratively and are particularly useful when analytical solutions are difficult or impossible to obtain.

2 Theory Behind the Methods

2.1 Bisection Method

The Bisection Method is a bracketing method that iteratively halves the search interval to approximate the root. Given an initial interval $[x_l, x_u]$ such that $f(x_l) \cdot f(x_u) < 0$, the midpoint is calculated as:

$$x_r = \frac{x_l + x_u}{2} \tag{1}$$

The function values at x_l and x_r are evaluated:

- If $f(x_l) \cdot f(x_r) < 0$, the root is in $[x_l, x_r]$, so x_u is updated.
- If $f(x_l) \cdot f(x_r) > 0$, the root is in $[x_r, x_u]$, so x_l is updated.
- If $f(x_r) = 0$, the exact root is found.

The process continues until the estimated error ε_a falls below a predefined threshold ε_s.

2.2 False-Position Method

The False-Position Method (or Regula Falsi) improves the root approximation by using a secant line to estimate the root:

$$x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)} \tag{2}$$

The function values determine how the interval is updated, similar to the Bisection Method. To prevent slow convergence, a relaxation technique is applied by adjusting $f(x_l)$ or $f(x_u)$ after multiple iterations without a sign change.

3 Java Implementation

The Java program implements both methods and allows switching between them using a switch-case statement.

3.1 Code Implementation

Listing 1: Java Code for Root-Finding

```
import java.util.function.Function;

public class SwitchingCalculation {
    public static double bisection(Function<Double, Double> f, double xl,
    double xu, double es, int imax) {
        int iter = 0;
        double xr = 0, xrold, ea = 100;
        while (iter < imax && ea > es) {
            xrold = xr;
            xr = (xl + xu) / 2;
            iter++;
            if (xr != 0) {
                ea = Math.abs((xr - xrold) / xr) * 100;
            }
            if (f.apply(xl) * f.apply(xr) < 0) xu = xr;
            else xl = xr;
        }
        return xr;
    }
    public static double falsePosition(Function<Double, Double> f, double xl
    double xu, double es, int imax) {
        int iter = 0, il = 0, iu = 0;
        double xr = 0, xrold, ea = 100;
        double fl = f.apply(xl), fu = f.apply(xu);
        while (iter < imax && ea > es) {
            xrold = xr;
            xr = xu - fu * (xl - xu) / (fl - fu);
            iter++;
            if (xr != 0) {
                ea = Math.abs((xr - xrold) / xr) * 100;
            }
            double fr = f.apply(xr);
            if (fl * fr < 0) xu = xr;
            else xl = xr;
        }
        return xr;
    }
}
```

3.2 Function Execution

The function is tested using the polynomial:

$$f(x) = -0.6x^2 + 2.4x + 5.5 \tag{3}$$

The analytical root is computed using the quadratic formula for comparison.

4 Results and Conclusion

Both methods provide iterative approximations of the root. The Bisection Method is reliable but converges slowly. The False-Position Method often converges faster but may stagnate if one bound remains unchanged for multiple iterations. The results align with the analytical solution, demonstrating the effectiveness of both approaches.