**Title**: Polynomial Root Finder with Bisection and False Position Methods

**Bisection Method:**

The Bisection method, often referred to as the binary search method, is a simple yet powerful approach for finding the roots of real-valued functions. Its theoretical foundation lies in the Intermediate Value Theorem, which states that a continuous function changing sign over an interval must have at least one root within that interval. The Bisection method operates as follows:

Start with an initial interval [a, b] where the function changes sign, meaning f(a) \* f(b) is negative.

Calculate the midpoint, c = (a + b) / 2.

Examine the sign of f(c):

If f(c) is close to zero (within a predefined tolerance), c is considered the root.

If the signs of f(c) and f(a) are the same, replace 'a' with 'c'; otherwise, replace 'b' with 'c'.

Repeat this process iteratively until the interval becomes sufficiently small, or the function value is close to zero.

**False Position Method (Regula Falsi):**

The False Position method, also known as the linear interpolation method, is another numerical technique used for finding roots of real-valued functions. The method's foundation is rooted in linear interpolation. It proceeds as follows:

Start with an initial interval [a, b] where the function changes sign (i.e., f(a) \* f(b) is negative).

Calculate the point 'c' where the line connecting (a, f(a)) and (b, f(b)) intersects the x-axis.

Examine the sign of f(c):

If f(c) is close to zero (within a predefined tolerance), 'c' is considered the root.

If the signs of f(c) and f(a) are the same, replace 'a' with 'c'; otherwise, replace 'b' with 'c'.

Repeat this process iteratively until the interval becomes sufficiently small or the function value is close to zero.

**Program:**

#include <iostream>

#include <cmath>

#include <iomanip>

using namespace std;

double evaluate(double coeffs[], int deg, double x)

{

    double res = 0;

    for (int i = deg; i >= 0; i--)

        res += coeffs[i] \* pow(x, i);

    return res;

}

void bisection(double coeffs[], int deg, double a, double b)

{

    double tol = 1e-4;

    int maxIter = 100;

    if (evaluate(coeffs, deg, a) \* evaluate(coeffs, deg, b) > 0)

    {

        cout << "Cannot apply bisection method to this interval." << endl;

        return;

    }

    cout << "Bisection Method:" << endl

         << setw(5) << "Step" << setw(12) << "a" << setw(12) << "b" << setw(12) << "x" << setw(12) << "f(x)" << setw(12) << "Error" << endl;

    for (int step = 1; step <= maxIter; step++)

    {

        double x = (a + b) / 2, fx = evaluate(coeffs, deg, x), prev;

        cout << setw(5) << step << setw(12) << a << setw(12) << b << setw(12) << x << setw(12) << fx << setw(12) << abs(x - prev) << endl;

        if (abs(x - prev) < tol)

        {

            cout << "Root found: " << setprecision(5) << x << endl;

            return;

        }

        prev = x;

        if (fx \* evaluate(coeffs, deg, a) < 0)

            b = x;

        else

            a = x;

    }

    cout << "Maximum iterations reached. No root found within the specified tolerance." << endl;

}

void falsePosition(double coeffs[], int deg, double a, double b)

{

    double tol = 1e-4;

    int maxIter = 100;

    if (evaluate(coeffs, deg, a) \* evaluate(coeffs, deg, b) > 0)

    {

        cout << "Cannot apply false position method to this interval." << endl;

        return;

    }

    cout << "False Position Method:" << endl

         << setw(5) << "Step" << setw(12) << "a" << setw(12) << "b" << setw(12) << "x" << setw(12) << "f(x)" << setw(12) << "Error" << endl;

    for (int step = 1; step <= maxIter; step++)

    {

        double fa = evaluate(coeffs, deg, a), fb = evaluate(coeffs, deg, b), prev;

        double x = (a \* fb - b \* fa) / (fb - fa);

        double fx = evaluate(coeffs, deg, x);

        cout << setw(5) << step << setw(12) << a << setw(12) << b << setw(12) << x << setw(12) << fx << setw(12) << abs(x - prev) << endl;

        if (abs(x - prev) < tol)

        {

            cout << "Root found: " << setprecision(5) << x << endl;

            return;

        }

        prev = x;

        if (fa \* fx < 0)

            b = x;

        else

            a = x;

    }

    cout << "Maximum iterations reached. No root found within the specified tolerance." << endl;

}

int main()

{

    int deg;

    double a, b;

    cout << "Enter degree of the polynomial: ";

    cin >> deg;

    cout << "Enter interval [a, b]: ";

    cin >> a >> b;

    double c[deg + 1];

    cout << "Enter coefficients from highest to lowest degree: ";

    for (int i = deg; i >= 0; i--)

        cin >> c[i];

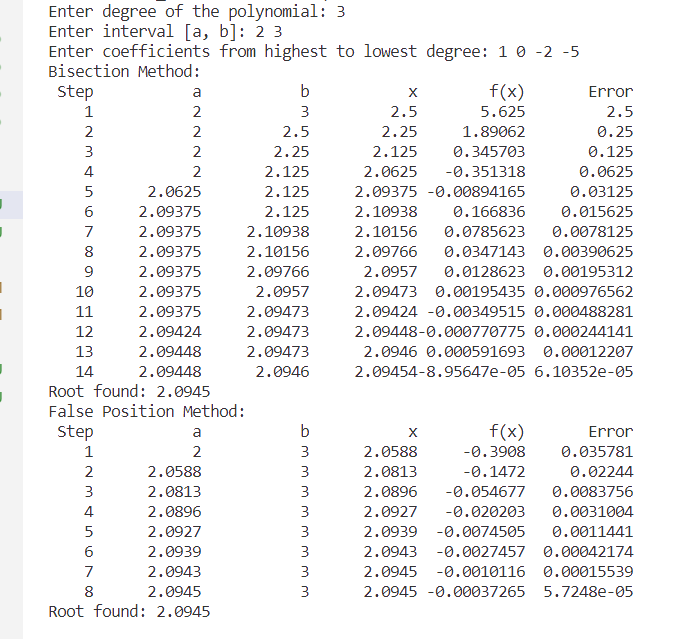
    bisection(c, deg, a, b);

    falsePosition(c, deg, a, b);

    return 0;

}

**Input & Output:**

****

**Discussion:** The Bisection and False Position methods provide reliable and straightforward solutions for finding roots of single-variable polynomial equations. Their application is not limited to polynomials, as they can be used for various continuous real-valued functions. The choice of method depends on the specific characteristics of the function and the desired trade-off between computational complexity and convergence speed. Understanding these methods is fundamental in solving numerical problems in mathematics, engineering, and other fields where polynomial root finding is a common task.