Numerical Methods Implementation and Performance Analysis

Introduction:

Numerical methods are fundamental tools in scientific computing and engineering, providing solutions to problems that may not be solvable by analytical means. These methods allow for the approximation of solutions to linear and nonlinear equations, which are common in many real-world applications. This assignment aims to implement various numerical methods, compare their performance, and analyze their effectiveness in solving both linear and nonlinear equations.

Methods:

Linear Equations:

- Gaussian Elimination: This method systematically reduces a system of linear equations to upper triangular form, allowing for straightforward back substitution to find the solutions.
- LU Decomposition: The LU Decomposition method involves decomposing the coefficient matrix into a product of a lower triangular matrix (L) and an upper triangular matrix (U). The solutions are then found by solving two simpler triangular systems.
- Jacobi Method: An iterative method that approximates the solution by repeatedly solving for each variable in the system based on an initial guess.
- Gauss-Seidel Method: Similar to the Jacobi method but with improved convergence properties, the Gauss-Seidel method uses the most recently updated values in each iteration.

Non-Linear Equations:

- Bisection Method: A root-finding method that repeatedly bisects an interval and selects the subinterval in which the root lies. It is a reliable but slow method.
- Newton-Raphson Method: A faster, iterative method that uses the derivative of the function to converge to the root, given a good initial guess.
- Secant Method: An approximation of the Newton-Raphson method that does not require the calculation of derivatives, making it more general but sometimes less stable.
- Fixed Point Iteration: A simple iterative method that rewrites the equation in the form $x=g(x)x = g(x)x=g(x)$ and iterates until convergence.

Numerical Methods Implementation and Performance Analysis

Approach:

Linear Equations:

- Gaussian Elimination: Study the process of transforming a matrix into upper triangular form and back substitution.
- LU Decomposition: Learn how to decompose a matrix into lower (L) and upper (U) triangular matrices.
- Jacobi Method: Understand the iterative approach and how to form successive approximations.
- Gauss-Seidel Method: Know the differences from the Jacobi method, particularly the use of updated values within iterations.

Non-Linear Equations:

- Bisection Method: Review the interval halving approach and its linear convergence.
- Newton-Raphson Method: Understand the iterative method involving function derivatives and its quadratic convergence.
- Secant Method: Study this approximation method based on secant lines.
- Fixed Point Iteration: Learn how to rewrite an equation in the form $x=g(x)x = g(x)x=g(x)$ and iterate to find a solution.

Code:

```
#include <iostream>
#include <cmath>
#include <iomanip>
#include <cstdlib>
using namespace std;

double f(double x) {
   return x * x * x - 6 * x;
}

double func1(double x) {
   return cos(x) - 3 * x + 1;
}
```

```
double g(double x) {
   return (1 + cos(x)) / 3;
}

double func2(double x) {
   return x * x * x - 4 * x - 9;
}

double derivFunc(double x) {
   return 3 * x * x - 4;
}

double func3(double x) {
   return x * x * x - x * x + 2;
}

void bisection(double a, double b, int n) {
   if (f(a) * f(b) >= 0) {
      cout << "Function does not change sign over the interval." << endl;
      return;
   }

   int i = 1;
   double root = 0;
   while (i <= n) {
      double x = (a + b) / 2;
      if (f(x) * f(b) < 0)
         a = x;
      else
         b = x;

      cout << "Iteration " << i << " x = " << x << " f(x) = " << f(x) << endl;

      if (fabs(f(x)) < 1e-6) {
         root = x;
         break;
      }
```

```
        i++;
    }
    cout << "Root is: " << root << endl;
}

void fixedPointIteration(double x0, double e, int N) {
    int step = 1;
    double x1;

    cout << setprecision(6) << fixed;
    cout << "Fixed Point Iteration Method" << endl;

    do {
        x1 = g(x0);
        cout << "Iteration-" << step << ":\t x1 = " << setw(10) << x1 << " and f(x1) =
" << setw(10) << func1(x1) << endl;

        if (fabs(func1(x1)) < e) {
            cout << "Root is " << x1 << endl;
            return;
        }

        x0 = x1;
        step++;

        if (step > N) {
            cout << "Not Convergent." << endl;
            return;
        }
    } while (fabs(func1(x1)) > e);
}

void newtonRaphson(double x) {
    double h;
    do {
        h = func2(x) / derivFunc(x);
        x = x - h;
```

```
    } while (fabs(h) >= 0.001);

    cout << "The value of the root is: " << x << endl;
}

void regulaFalsi(double a, double b) {
    if (func3(a) * func3(b) >= 0) {
        cout << "Function does not change sign over the interval." << endl;
        return;
    }

    double c;
    for (int i = 0; i < 1000000; i++) {
        c = (a * func3(b) - b * func3(a)) / (func3(b) - func3(a));

        if (func3(c) == 0)
            break;
        else if (func3(c) * func3(a) < 0)
            b = c;
        else
            a = c;
    }
    cout << "The value of the root is: " << c << endl;
}

void gaussianElimination(int n, double a[10][10], double b[10]) {
    double augmented[10][11];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            augmented[i][j] = a[i][j];
        }
        augmented[i][n] = b[i];
    }

    for (int k = 0; k < n; k++) {
        for (int i = k + 1; i < n; i++) {
            if (fabs(augmented[i][k]) > fabs(augmented[k][k])) {
```

```
            for (int j = 0; j <= n; j++) {
                swap(augmented[k][j], augmented[i][j]);
            }
        }
    }

    for (int i = k + 1; i < n; i++) {
        double factor = augmented[i][k] / augmented[k][k];
        for (int j = k; j <= n; j++) {
            augmented[i][j] -= factor * augmented[k][j];
        }
    }
  }

  double x[10];
  for (int i = n - 1; i >= 0; i--) {
    x[i] = augmented[i][n] / augmented[i][i];
    for (int j = i - 1; j >= 0; j--) {
        augmented[j][n] -= augmented[j][i] * x[i];
    }
  }

  cout << "Gaussian Elimination Method" << endl;
  for (int i = 0; i < n; i++) {
    cout << "x[" << i + 1 << "] = " << x[i] << endl;
  }
}

void jacobiMethod(int n, double a[10][10], double b[10], double x[10], int
maxIter, double tol) {
  double x_new[10];
  int iter = 0;
  while (iter < maxIter) {
    for (int i = 0; i < n; i++) {
        x_new[i] = b[i];
        for (int j = 0; j < n; j++) {
            if (i != j)
```

```
            x_new[i] -= a[i][j] * x[j];
        }
        x_new[i] /= a[i][i];
    }

    bool converge = true;
    for (int i = 0; i < n; i++) {
        if (fabs(x_new[i] - x[i]) > tol)
            converge = false;
        x[i] = x_new[i];
    }

    if (converge)
        break;

    iter++;
  }

  cout << "Jacobi Method" << endl;
  for (int i = 0; i < n; i++) {
    cout << "x" << i + 1 << " = " << x[i] << endl;
  }
  cout << "Iterations: " << iter << endl;
}

void gaussSeidelMethod(int n, double a[10][10], double b[10], double x[10], int
maxIter, double tol) {
  int iter = 0;
  while (iter < maxIter) {
    bool converge = true;
    for (int i = 0; i < n; i++) {
        double sum = b[i];
        for (int j = 0; j < n; j++) {
            if (i != j)
                sum -= a[i][j] * x[j];
        }
        double x_new = sum / a[i][i];
```

```
        if (fabs(x_new - x[i]) > tol)
            converge = false;
        x[i] = x_new;
      }

      if (converge)
        break;

      iter++;
    }

    cout << "Gauss-Seidel Method" << endl;
    for (int i = 0; i < n; i++) {
      cout << "x" << i + 1 << " = " << x[i] << endl;
    }
    cout << "Iterations: " << iter << endl;
}

int main() {
    int choice, problemType;
    bool keepRunning = true;
    cout << "Choose the type of problem:\n";
    cout << "1. Linear Equations\n2. Non-Linear Equations\n";
    cout << "Enter your choice: ";
    cin >> problemType;

    while (keepRunning) {
      if (problemType == 1) {
        cout << "Choose a linear method to solve the equations:\n";
                cout << "1. Gaussian Elimination Method\n2. Jacobi Method\n3.
Gauss-Seidel Method\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
          case 1: {
            int n;
```

```
    double a[10][10], b[10];
    cout << "Enter number of equations: ";
    cin >> n;
    cout << "Enter the coefficients matrix (A):" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> a[i][j];
        }
    }
    cout << "Enter the constant terms vector (B):" << endl;
    for (int i = 0; i < n; i++) {
        cin >> b[i];
    }
    gaussianElimination(n, a, b);
    break;
}
case 2: {
    int n, maxIter;
    double a[10][10], b[10], x[10], tol;
    cout << "Enter number of equations: ";
    cin >> n;
    cout << "Enter the coefficients matrix (A):" << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> a[i][j];
        }
    }
    cout << "Enter the constant terms vector (B):" << endl;
    for (int i = 0; i < n; i++) {
        cin >> b[i];
        x[i] = 0;
    }
    cout << "Enter maximum iterations: ";
    cin >> maxIter;
    cout << "Enter tolerance: ";
    cin >> tol;
    jacobiMethod(n, a, b, x, maxIter, tol);
```

```
            break;
        }
        case 3: {
            int n, maxIter;
            double a[10][10], b[10], x[10], tol;
            cout << "Enter number of equations: ";
            cin >> n;
            cout << "Enter the coefficients matrix (A):" << endl;
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    cin >> a[i][j];
                }
            }
            cout << "Enter the constant terms vector (B):" << endl;
            for (int i = 0; i < n; i++) {
                cin >> b[i];
                x[i] = 0;
            }
            cout << "Enter maximum iterations: ";
            cin >> maxIter;
            cout << "Enter tolerance: ";
            cin >> tol;
            gaussSeidelMethod(n, a, b, x, maxIter, tol);
            break;
        }
        default:
            cout << "Invalid choice!" << endl;
            keepRunning = false;
            break;
    }
} else if (problemType == 2) {
    cout << "Choose a non-linear method to find the root:\n";
        cout << "1. Bisection Method\n2. Fixed Point Iteration\n3. Newton-Raphson Method\n4. Regula Falsi Method\n";
    cout << "Enter your choice: ";
    cin >> choice;
```

```
switch (choice) {
    case 1: {
        double a, b;
        int n;
        cout << "Enter interval [a, b]: ";
        cin >> a >> b;
        cout << "Enter number of iterations: ";
        cin >> n;
        bisection(a, b, n);
        break;
    }
    case 2: {
        double x0, e;
        int N;
        cout << "Enter initial guess: ";
        cin >> x0;
        cout << "Enter tolerable error: ";
        cin >> e;
        cout << "Enter maximum iterations: ";
        cin >> N;
        fixedPointIteration(x0, e, N);
        break;
    }
    case 3: {
        double x0;
        cout << "Enter initial guess: ";
        cin >> x0;
        newtonRaphson(x0);
        break;
```

```
            }
            case 4: {
                double a, b;
                cout << "Enter interval [a, b]: ";
                cin >> a >> b;
                regulaFalsi(a, b);
                break;
            }
            default:
                cout << "Invalid choice!" << endl;
                keepRunning = false;
                break;
        }
    } else {
        cout << "Invalid problem type choice!" << endl;
        keepRunning = false;
    }
}

return 0;
}
```

Numerical Methods Implementation and Performance Analysis

Output:

```
Choose the type of problem:
1. Linear Equations
2. Non-Linear Equations
Enter your choice: 1
Choose a linear method to solve the equations:
1. Gaussian Elimination Method
2. Jacobi Method
3. Gauss-Seidel Method
Enter your choice: 1
Enter number of equations: 2
Enter the coefficients matrix (A):
2
3
4
5
Enter the constant terms vector (B):
2
3
Gaussian Elimination Method
x[1] = -0.5
x[2] = 1
Choose a linear method to solve the equations:
1. Gaussian Elimination Method
2. Jacobi Method
3. Gauss-Seidel Method
Enter your choice:
```

```
Choose the type of problem:
1. Linear Equations
2. Non-Linear Equations
Enter your choice: 2
Choose a non-linear method to find the root:
1. Bisection Method
2. Fixed Point Iteration
3. Newton-Raphson Method
4. Regula Falsi Method
Enter your choice: 1
Enter interval [a, b]: 2 3
Enter number of iterations: 5
Iteration 1 x = 2.5 f(x) = 0.625
Iteration 2 x = 2.25 f(x) = -2.10938
Iteration 3 x = 2.375 f(x) = -0.853516
Iteration 4 x = 2.4375 f(x) = -0.142822
Iteration 5 x = 2.46875 f(x) = 0.233856
Root is: 0
Choose a non-linear method to find the root:
1. Bisection Method
2. Fixed Point Iteration
3. Newton-Raphson Method
4. Regula Falsi Method
Enter your choice:
```

Numerical Methods Implementation and Performance Analysis

It is the output sample of my code. I can select the Linear or Non linear method first then I can select the method under these two types.

Performance Analysis:

Linear Equation

Method	Convergence	Time Complexity	Iterations Required	Execution Time
Gaussian Elimination	Direct (exact)	O(N^3)	Direct Method	High
LU Decomposition	Direct(Exact)	O(N^2)	Direct Method	Moderate to High
Jacobi Method	Linear(Slow)	O(N^2) to O(N^3) Per Iteration	High	Moderate to High
Gauss-Seidel Method	Linear(Moderate)	O(N^2) to O(N^3) Per Iteration	Moderate	Moderate

Non-Linear Equation

Method	Convergence	Time Complexity	Iterations Required	Execution Time
Bisection	Linear(Slow)	O(Log N)	Moderate	Moderate
Newton-Raphson	Quadratic(fast)	O(Log N) per iteration	Low	Low
Secant	Super Linear(Moderate)	O(Log N) per iteration	Moderate	Moderate
Fixed Point Iteration	Linear(Slow)	O(N)	High	Moderate to High

Accuracy:

General Accuracy: The code generally implements standard methods for solving linear and nonlinear equations. Each method has its strengths and limitations.

Precision Issues: Floating-point arithmetic can affect accuracy. Ensuring numerical stability through techniques like pivoting, adjusting tolerance, and proper parameter selection is crucial.

Method Choice: The choice of method should align with the problem characteristics (matrix condition for linear systems, function properties for nonlinear systems) to ensure accurate results.

Numerical Methods Implementation and Performance Analysis

Summary:

The C++ code provides a range of numerical methods for solving linear and nonlinear equations:

1. Linear Equation Methods:
    - **Gaussian Elimination:** A direct method for solving systems of linear equations by transforming the system into upper triangular form and then performing back-substitution. It's efficient for small to moderately sized systems.
    - **Jacobi Method:** An iterative method suitable for large, sparse systems with a diagonally dominant matrix. It updates each variable independently in each iteration.
    - **Gauss-Seidel Method:** Similar to the Jacobi Method but updates variables sequentially, often resulting in faster convergence.
2. Non-Linear Equation Methods:
    - **Bisection Method:** A robust, bracketing method that finds roots by repeatedly narrowing the interval where the function changes sign. It guarantees convergence but may be slow.
    - **Fixed Point Iteration:** An iterative method that requires transforming the equation into a suitable fixed point form. Convergence depends on the function's properties and initial guess.
    - **Newton-Raphson Method:** An iterative method that uses the function's derivative to rapidly converge to a root. It requires a good initial guess and the computation of derivatives.
    - **Regula Falsi Method:** A combination of the bisection method and the secant method that improves convergence speed while maintaining robustness.