IN THE NAME OF ALLAH, THE GREATEST THE MOST MERCIFUL

INTERNATIONAL ISLAMIC UNIVERSITY CHITTAGONG

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Project Report

Matric / ID No.	**:** C211032
ID No. (in words)	**:** C-Two-One-One- Zero-Three-Two
Name	**:** Rashedul Arefin Ifty
Semester	**:** 7th
Section	**:** 7AM
Course Code	**:** CSE-4746
Course Title	**:** Numerical Methods Lab
Course Teacher Name	**:** Prof. Mohammed Shamsul Alam
Email ID	**:** r.a.ifty2001@gmail.com
Contact Number	**:** 01840003222

Submitted To:

Prof. Mohammed Shamsul Alam
Professor, Dept. of CSE
International Islamic University Chittagong

Resource Allocation Using System of Linear Equations

Introduction: In contemporary project management, the effective allocation of resources is crucial for achieving project goals efficiently and within constraints. These constraints often include limited manpower, specific equipment availability, and strict timeframes. The ability to accurately quantify and allocate these resources across various project tasks is not just a logistical challenge but a mathematical problem that can be effectively addressed using systems of linear equations.

Problem Statement: The project aims to tackle the challenge of resource allocation using mathematical methods based on systems of linear equations. Specifically, it focuses on scenarios where a project manager needs to determine the exact quantities of manpower, equipment, and time required for each task or phase of a project. This determination must be made under given constraints and objectives, such as minimizing costs, optimizing resource utilization, or meeting project deadlines.

Objectives:

- **Resource Quantification:** Develop computational models that can accurately calculate the quantities of manpower, equipment, and time needed based on input parameters such as task requirements and resource availability.

- **Method Evaluation:** Compare and analyze different mathematical methods used to solve systems of linear equations, including Matrix Inversion, Cramer's Rule, and iterative methods like Jacobi and Gauss-Seidel. Assess their suitability and efficiency in various project scenarios.

- **Accuracy Assessment:** Implement mechanisms to evaluate the accuracy and reliability of each method's results compared to expected outcomes. This includes calculating error margins and determining the conditions under which each method performs optimally.

- **User Interface Design:** Design an intuitive user interface that allows project managers or stakeholders to input task requirements, constraints, and other relevant data. The interface should facilitate the selection of solution methods and interpretation of results in a clear and accessible manner.

Features:

- User input for matrix coefficients and constants.
- Matrix Inversion Method for direct solution.
- Cramer's Rule for determinant-based solution.
- Jacobi and Gauss-Seidel methods for iterative solutions.
- Accuracy assessment against expected results.
- Clear menu-driven interface for method selection.
- Efficient computation balancing accuracy and performance.

Implementation Overview:

1. Input Handling
- Function: `getInput()`
- Description: Accepts user input for coefficients of the matrix and constants vector.

2. Matrix Inversion Method
- Function: `matrixInversionMethod(double x[N])`
- Description: Computes solutions using matrix inversion, ensuring the invertibility of the matrix before proceeding.

3. Cramer's Rule
- Function: `cramersRule(double x[N])`
- Description: Implements a method based on determinants to compute solutions for each variable.

4. Jacobi Method
- Function: `jacobiMethod(double x[N])`
- Description: Iteratively refines solutions by updating each variable independently until convergence.

5. Gauss-Seidel Method
- Function: `gaussSeidelMethod(double x[N])`
- Description: Improves upon the Jacobi method by using updated values immediately within the same iteration.

6. Accuracy Calculation
- Function: `calculateAccuracy(double calculated[N], double expected[N], double accuracy[N])`
- Description: Computes accuracy by comparing computed results with expected outcomes, determining deviations for assessment.

7. User Interface
- Function: `main()`
- Description: Provides a menu-driven interface for users to select solution methods, input coefficients and constants, and view results.

8. Error Handling
- Description: Includes checks to handle scenarios where the matrix is not invertible, ensuring robustness and providing appropriate feedback.

9. Performance Considerations
- Description: Optimizes iterative methods to balance computational efficiency and convergence accuracy, ensuring effective resource allocation predictions.

10. Modular Code Structure
- Description: Organizes code into reusable functions for clarity and maintainability, facilitating easy debugging and future enhancements.

Workflow in Coding:

1. **Input Acquisition:** Users input coefficients of the matrix and constants vector.
2. **Menu Selection:** Users choose from methods including Matrix Inversion, Cramer's Rule, Jacobi Method, Gauss-Seidel Method, and accuracy check.
3. **Method Execution:** Selected method computes solutions or accuracy based on user-provided inputs.

4. **Output Display:** Results are displayed, including computed resource quantities and accuracy assessments.
5. **Error Handling:** Checks for scenarios where the matrix is not invertible, providing feedback to ensure reliable computation.

Problem: Suppose a project manager is allocating resources across three tasks, each requiring different amounts of manpower, equipment, and time. The goal is to determine how many units of each resource are needed to complete the project under given constraints.

Dataset:

Eq. 1: $27x + 6y – z = 85$	**Eq. 2: $6x + 15y + 2z = 72$**	**Eq. 3: $x + y + 54z = 110$**
x: Units of resource 1 (e.g., manpower) **y:** Units of resource 2 (e.g., equipment) **z:** Units of resource 3 (e.g., time) **27:** Resource 1 coefficient **6:** Resource 2 coefficient **−1:** Resource 3 coefficient **85:** Total resource requirement	**x:** Units of resource 1 (e.g., manpower) **y:** Units of resource 2 (e.g., equipment) **z:** Units of resource 3 (e.g., time) **6:** Resource 1 coefficient **15:** Resource 2 coefficient **2:** Resource 3 coefficient **72:** Total resource requirement	**x:** Units of resource 1 (e.g., manpower) **y:** Units of resource 2 (e.g., equipment) **z:** Units of resource 3 (e.g., time) **1:** Resource 1 coefficient **1:** Resource 2 coefficient **54:** Resource 3 coefficient **110:** Total resource requirement

Explanation:

- Equation 1: Represents the resource allocation equation where the left-hand side calculates the total resources required (manpower, equipment, and time) to complete Task 1, which equals 85 units.
- Equation 2: Represents the resource allocation eqn. for Task 2.
- Equation 3: Represents the resource allocation eqn. for Task 3.

Dataset Format:

Equation	Coefficients (A)	Constants (b)
1	[27, 6, -1]	85
2	[6, 15, 2]	72
3	[1, 1, 54]	110

- Equation: Identifier for each equation.
- Coefficients **(A)**: Coefficients of variables x, y, and z in the equation $Ax=b$.
- Constants **(b)**: The constant term on the right-hand side of the equation.

Result Analysis:

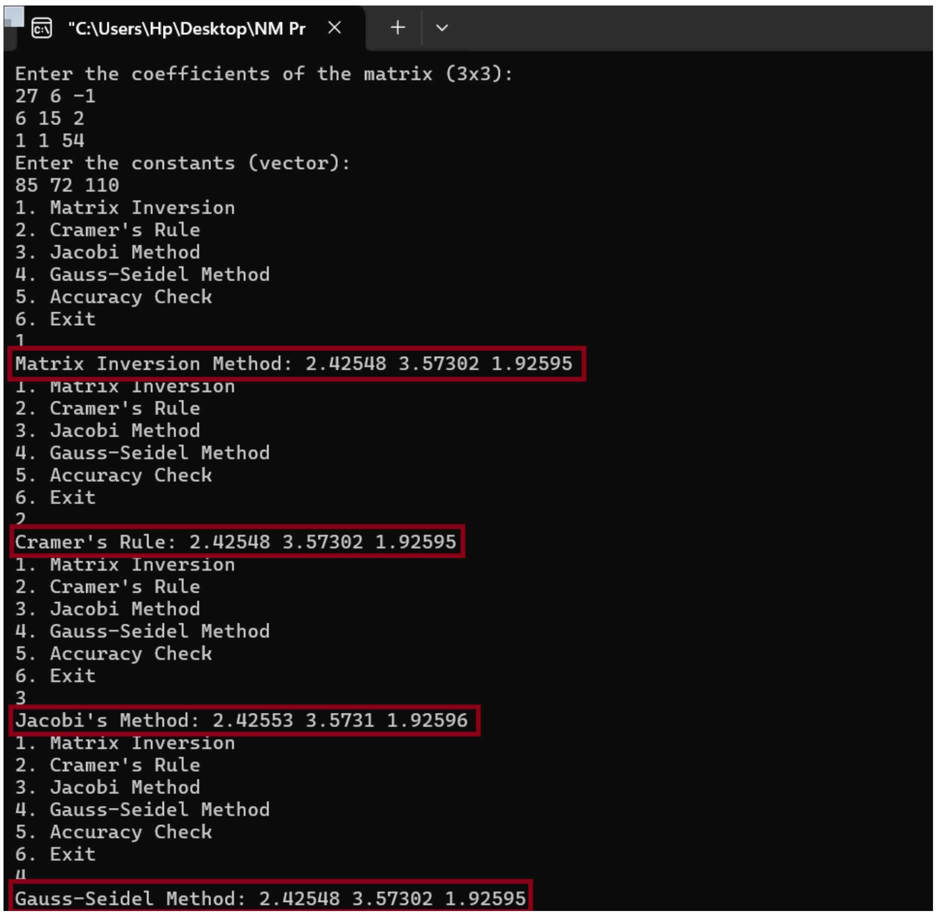
Methods\Outputs	x	y	z
Actual Solution	2.425476298	3.573025634	1.925953853
Metrix Inversion Method	2.42548	3.57302	1.92595
Cramer's Rule	2.42548	3.57302	1.92595
Jacobi's Method	2.42553	3.5731	1.92596
Gauss-Seidel Method	2.42548	3.57302	1.92595

Table 1: Solution Comparison

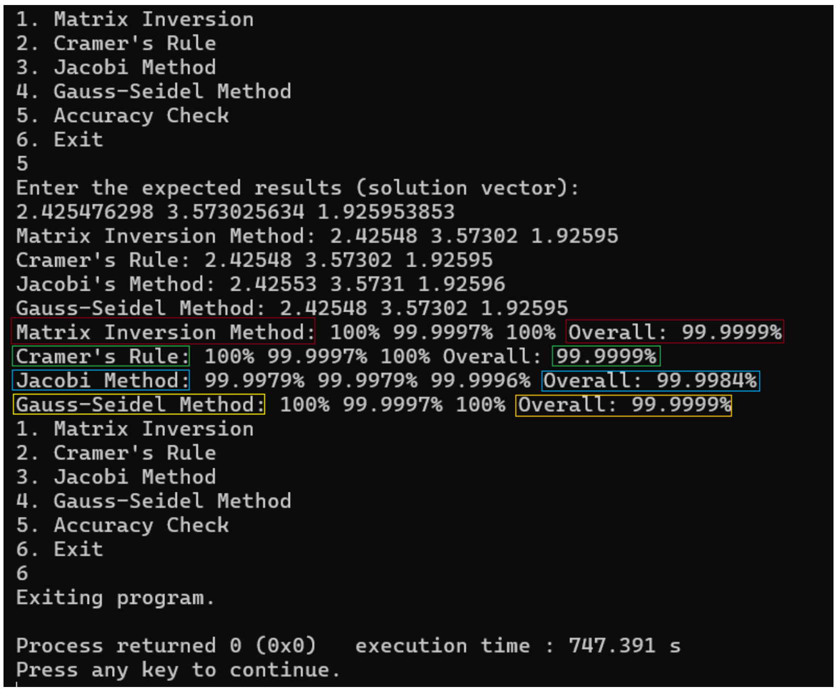
Methods\Outputs	x	y	z	Overall
Metrix Inversion Method	100%	99.9997%	100%	99.9999%
Cramer's Rule	100%	99.9997%	100%	99.9999%
Jacobi's Method	99.9979%	99.9979%	99.9996%	99.9984%
Gauss-Seidel Method	100%	99.9997%	100%	99.9999%

Table 2: Accuracy Comparison

Outputs:



```
Enter the coefficients of the matrix (3x3):
27 6 -1
6 15 2
1 1 54
Enter the constants (vector):
85 72 110
1. Matrix Inversion
2. Cramer's Rule
3. Jacobi Method
4. Gauss-Seidel Method
5. Accuracy Check
6. Exit
1
Matrix Inversion Method: 2.42548 3.57302 1.92595
1. Matrix Inversion
2. Cramer's Rule
3. Jacobi Method
4. Gauss-Seidel Method
5. Accuracy Check
6. Exit
2
Cramer's Rule: 2.42548 3.57302 1.92595
1. Matrix Inversion
2. Cramer's Rule
3. Jacobi Method
4. Gauss-Seidel Method
5. Accuracy Check
6. Exit
3
Jacobi's Method: 2.42553 3.5731 1.92596
1. Matrix Inversion
2. Cramer's Rule
3. Jacobi Method
4. Gauss-Seidel Method
5. Accuracy Check
6. Exit
4
Gauss-Seidel Method: 2.42548 3.57302 1.92595
```



```
1. Matrix Inversion
2. Cramer's Rule
3. Jacobi Method
4. Gauss-Seidel Method
5. Accuracy Check
6. Exit
5
Enter the expected results (solution vector):
2.425476298 3.573025634 1.925953853
Matrix Inversion Method: 2.42548 3.57302 1.92595
Cramer's Rule: 2.42548 3.57302 1.92595
Jacobi's Method: 2.42553 3.5731 1.92596
Gauss-Seidel Method: 2.42548 3.57302 1.92595
Matrix Inversion Method: 100% 99.9997% 100% Overall: 99.9999%
Cramer's Rule: 100% 99.9997% 100% Overall: 99.9999%
Jacobi Method: 99.9979% 99.9979% 99.9996% Overall: 99.9984%
Gauss-Seidel Method: 100% 99.9997% 100% Overall: 99.9999%
1. Matrix Inversion
2. Cramer's Rule
3. Jacobi Method
4. Gauss-Seidel Method
5. Accuracy Check
6. Exit
6
Exiting program.

Process returned 0 (0x0)   execution time : 747.391 s
Press any key to continue.
```

Source Code:

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 3;
double a[N][N];
double b[N];
double expectedResults[N];

// Function for Matrix Inversion Method
void matrixInversionMethod(double x[N])
{
    double A[N][N];
    double det_A = a[0][0] * (a[1][1] * a[2][2] - a[1][2] * a[2][1]) -
                   a[0][1] * (a[1][0] * a[2][2] - a[1][2] * a[2][0]) +
                   a[0][2] * (a[1][0] * a[2][1] - a[1][1] * a[2][0]);

    if (det_A == 0)
    {
        cout << "Matrix Inversion cannot be applied. Determinant is zero." << endl;
        return;
    }

    A[0][0] = (a[1][1] * a[2][2] - a[1][2] * a[2][1]) / det_A;
    A[0][1] = (a[0][2] * a[2][1] - a[0][1] * a[2][2]) / det_A;
    A[0][2] = (a[0][1] * a[1][2] - a[0][2] * a[1][1]) / det_A;
    A[1][0] = (a[1][2] * a[2][0] - a[1][0] * a[2][2]) / det_A;
    A[1][1] = (a[0][0] * a[2][2] - a[0][2] * a[2][0]) / det_A;
    A[1][2] = (a[0][2] * a[1][0] - a[0][0] * a[1][2]) / det_A;
    A[2][0] = (a[1][0] * a[2][1] - a[1][1] * a[2][0]) / det_A;
    A[2][1] = (a[0][1] * a[2][0] - a[0][0] * a[2][1]) / det_A;
    A[2][2] = (a[0][0] * a[1][1] - a[0][1] * a[1][0]) / det_A;

    for (int i = 0; i < N; i++)
    {
        x[i] = 0;
        for (int j = 0; j < N; j++)
        {
            x[i] += A[i][j] * b[j];
        }
    }
    cout << "Matrix Inversion Method: ";
    for (int i = 0; i < N; i++)
        cout << x[i] << " ";
    cout << endl;
}

// Function for Cramer's Rule
void cramersRule(double x[N])
{
    double tmp[N];
    double det = a[0][0] * (a[1][1] * a[2][2] - a[1][2] * a[2][1]) -
                 a[0][1] * (a[1][0] * a[2][2] - a[1][2] * a[2][0]) +
                 a[0][2] * (a[1][0] * a[2][1] - a[1][1] * a[2][0]);

    if (det == 0)
    {
        cout << "Cramer's Rule cannot be applied. Determinant is zero." << endl;
        return;
    }

    auto cd = [&](int row)
    {
        for (int col = 0; col < N; col++)
        {
            tmp[col] = a[col][row];
            a[col][row] = b[col];
        }

        double result = a[0][0] * (a[1][1] * a[2][2] - a[1][2] * a[2][1]) -
                        a[0][1] * (a[1][0] * a[2][2] - a[1][2] * a[2][0]) +
                        a[0][2] * (a[1][0] * a[2][1] - a[1][1] * a[2][0]);

        for (int col = 0; col < N; col++)
        {
            a[col][row] = tmp[col];
        }

        return result;
    };

    x[0] = cd(0) / det;
    x[1] = cd(1) / det;
    x[2] = cd(2) / det;

    cout << "Cramer's Rule: ";
    for (int i = 0; i < N; i++)
        cout << x[i] << " ";
    cout << endl;
}

// Function for Jacobi's Method
void jacobiMethod(double x[N])
{
    double preX, preY, preZ;
    x[0] = x[1] = x[2] = 0;
    do
    {
        preX = x[0];
        preY = x[1];
        preZ = x[2];
        x[0] = (b[0] - a[0][2] * preZ - a[0][1] * preY) / a[0][0];
        x[1] = (b[1] - a[1][2] * preZ - a[1][0] * preX) / a[1][1];
        x[2] = (b[2] - a[2][0] * preX - a[2][1] * preY) / a[2][2];
    }
    while (abs(preX - x[0]) > 0.001 || abs(preY - x[1]) > 0.001 || abs(preZ - x[2]) > 0.001);

    cout << "Jacobi's Method: ";
    for (int i = 0; i < N; i++)
        cout << x[i] << " ";
    cout << endl;
}

// Function for Gauss-Seidel Method
void gaussSeidelMethod(double x[N])
{
    x[0] = x[1] = x[2] = 0;
    int iterations = 10;
    while (iterations--)
    {
        x[0] = (b[0] - a[0][2] * x[2] - a[0][1] * x[1]) / a[0][0];
        x[1] = (b[1] - a[1][2] * x[2] - a[1][0] * x[0]) / a[1][1];
        x[2] = (b[2] - a[2][0] * x[0] - a[2][1] * x[1]) / a[2][2];
    }
    cout << "Gauss-Seidel Method: ";
    for (int i = 0; i < N; i++)
        cout << x[i] << " ";
    cout << endl;
}

// Function to calculate accuracy
void calculateAccuracy(double calculated[N], double expected[N], double accuracy[N])
{
    for (int i = 0; i < N; i++)
    {
        accuracy[i] = (abs(calculated[i] - expected[i]) / abs(expected[i])) * 100;
    }
}

// Function to get input
void getInput()
{
    cout << "Enter the coefficients of the matrix (" << N << "x" << N << "):" << endl;
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            cin >> a[i][j];
        }
    }
    cout << "Enter the constants (vector):" << endl;
    for (int i = 0; i < N; i++)
    {
        cin >> b[i];
    }
}

// Function to get expected results for accuracy check
void getExpectedResults()
{
    cout << "Enter the expected results (solution vector):" << endl;
    for (int i = 0; i < N; i++)
    {
        cin >> expectedResults[i];
    }
}

int main()
{
    getInput();

    int x;
    double results[N];
menu:
    cout << "1. Matrix Inversion" << endl;
    cout << "2. Cramer's Rule" << endl;
    cout << "3. Jacobi Method" << endl;
    cout << "4. Gauss-Seidel Method" << endl;
    cout << "5. Accuracy Check" << endl;
    cout << "6. Exit" << endl;

    cin >> x;

    switch (x)
    {
    case 1:
        matrixInversionMethod(results);
        break;
    case 2:
        cramersRule(results);
        break;
    case 3:
        jacobiMethod(results);
        break;
    case 4:
        gaussSeidelMethod(results);
        break;
    case 5:
    {
```

```
        getExpectedResults();
        double miResults[N], crResults[N], jmResults[N], gsResults[N];
        double miAccuracy[N], crAccuracy[N], jmAccuracy[N], gsAccuracy[N];

        matrixInversionMethod(miResults);
        cramersRule(crResults);
        jacobiMethod(jmResults);
        gaussSeidelMethod(gsResults);

        calculateAccuracy(miResults, expectedResults, miAccuracy);
        calculateAccuracy(crResults, expectedResults, crAccuracy);
        calculateAccuracy(jmResults, expectedResults, jmAccuracy);
        calculateAccuracy(gsResults, expectedResults, gsAccuracy);

        cout << "Matrix Inversion Method: ";
        for (int i = 0; i < N; i++)
        {
            cout << 100 - miAccuracy[i] << "% ";
        }
        cout << "Overall: " << 100 - (miAccuracy[0] + miAccuracy[1] + miAccuracy[2]) / 3 << "%"
<< endl;

        cout << "Cramer's Rule: ";
        for (int i = 0; i < N; i++)
        {
            cout << 100 - crAccuracy[i] << "% ";
        }
        cout << "Overall: " << 100 - (crAccuracy[0] + crAccuracy[1] + crAccuracy[2]) / 3 << "%"
<< endl;

        cout << "Jacobi Method: ";
        for (int i = 0; i < N; i++)
        {
            cout << 100 - jmAccuracy[i] << "% ";
        }
        cout << "Overall: " << 100 - (jmAccuracy[0] + jmAccuracy[1] + jmAccuracy[2]) / 3 << "%"
<< endl;

        cout << "Gauss-Seidel Method: ";
        for (int i = 0; i < N; i++)
        {
            cout << 100 - gsAccuracy[i] << "% ";
        }
        cout << "Overall: " << 100 - (gsAccuracy[0] + gsAccuracy[1] + gsAccuracy[2]) / 3 << "%"
<< endl;

        break;
    }
    case 6:
        cout << "Exiting program." << endl;
        return 0;
    default:
        cout << "Invalid option. Please try again." << endl;
    }

    goto menu;

    return 0;
}
```

Future Plan and Possible Extension: For future plans and possible extensions of this project on solving systems of linear equations using numerical methods, consider integrating advanced algorithms like QR decomposition and iterative methods, enhancing input handling for larger matrices, implementing error analysis for stability, developing visualization tools for result interpretation, exploring parallel computing for performance optimization, and potentially extending the application to solve optimization problems or support machine learning applications, all while ensuring a modular design, thorough documentation, and rigorous testing for reliability and usability across different domains and user scenarios.

Conclusion: Expanding the project on solving systems of linear equations using numerical methods presents opportunities to integrate advanced algorithms, enhance input capabilities, implement error analysis, develop visualization tools, explore parallel computing, extend into optimization and machine learning domains, and ensure a robust, user-friendly design through modular architecture, comprehensive documentation, and rigorous testing. These steps will not only enhance the project's functionality and performance but also broaden its applicability across diverse computational tasks and educational contexts.
