REPORT : Gauss–Jordan Linear Equation Solver Using Python

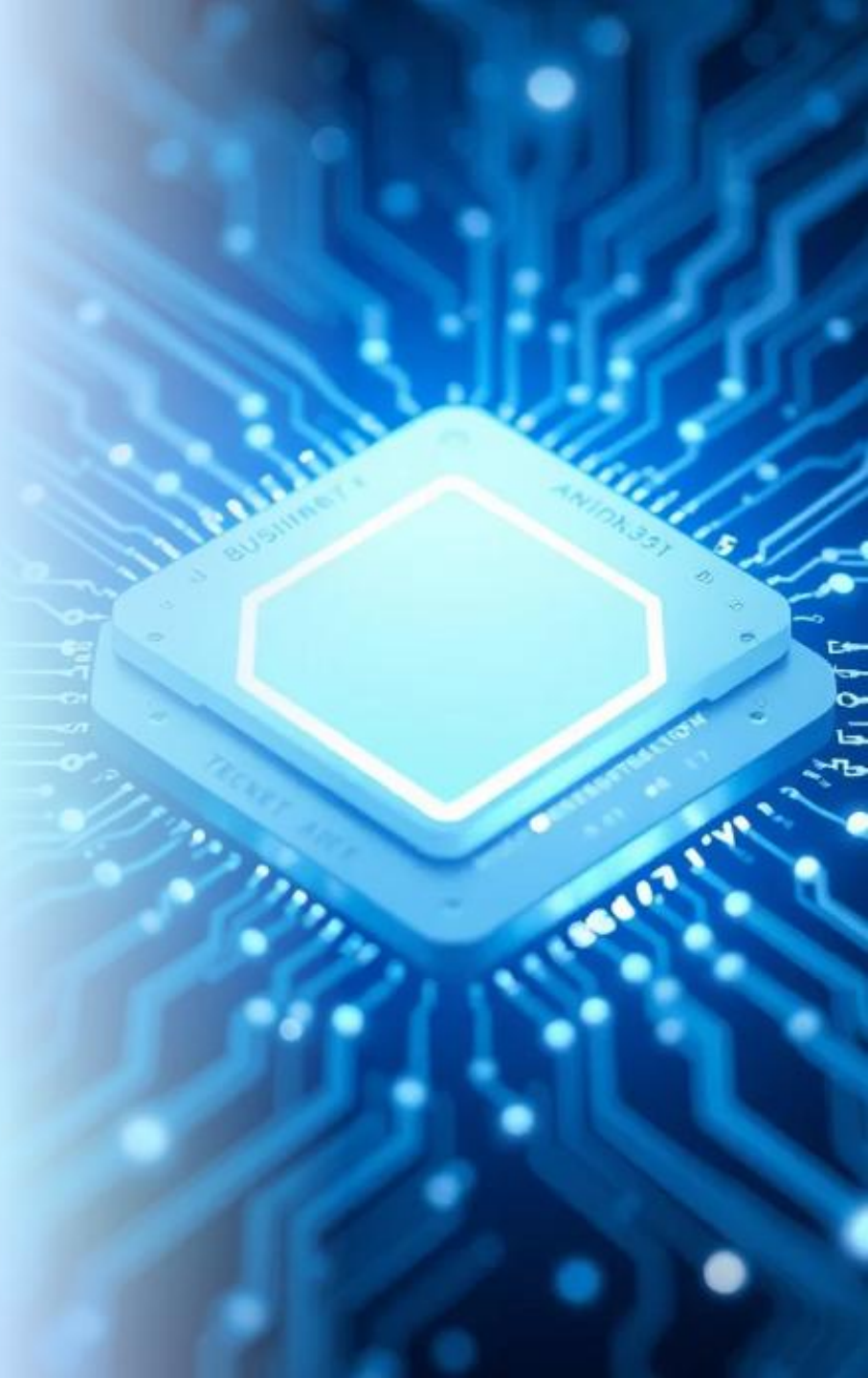
By : Divya Suyash Nishad (25BCE11077)

CONTENTS

1. Problem Statement

Solving systems of linear equations is a fundamental task in mathematics, engineering, and computer science. Manually solving large systems is time-consuming and prone to errors. Therefore, there is a need for a computerized solution that can take an augmented matrix as input and automatically compute the values of unknown variables accurately.

This project addresses the real-world requirement of solving simultaneous linear equations using an efficient numerical method — Gauss–Jordan Elimination.



2. Real-World Problem / Need Identified

Many scientific, engineering, and economic problems require solving linear systems, such as:

- Network flow analysis
- Electrical circuit analysis
- Statistical regression
- Computer graphics transformations
- Physics and mechanics problems

A digital solver helps automate these calculations and eliminates human error.

3. Objective / Expected Outcomes

The objectives of this project are:

1. To design a Python program that accepts a user-defined augmented matrix.
2. To solve the system using Gauss–Jordan Elimination.
3. To provide accurate values of the unknown variables.
4. To demonstrate programming concepts learned in the course.
5. To understand matrix transformations and numerical computation.
6. To give users a clean interface, error handling, and clear output.

Expected outcomes:

1. A correct Reduced Row Echelon Form (RREF) matrix.
2. Correct solution for each variable.
3. Smooth user interaction and logically structured code.

4. Concepts Applied from the Course

This project uses several core programming concepts:

Input Handling

Accepting user input for rows, columns, and matrix values.

Typecasting

Converting input strings to int or float.

Loops

- for loops for iterating through matrix rows and columns.
- while loops for repeated input validation.

Nested Loops

Used to build the matrix row-by-row and column-by-column.

Conditional Statements (if-else)

Checking pivot values, handling row swapping, and validating input.

Functions

Modular approach using:

- get_matrix()
- gauss_jordan()
- print_matrix()
- print_solution()

Arrays / NumPy

Using NumPy arrays for mathematical operations and matrix manipulation.

Exception Handling

Using try-except to catch invalid user inputs.

Mathematical Algorithm Implementation

Normalized row operations, pivoting, and elimination logic.

5. Appropriate Tools, Libraries, and Techniques

1

Python

selected for simplicity and readability.

2

NumPy

chosen for efficient matrix operations and convenience.

3

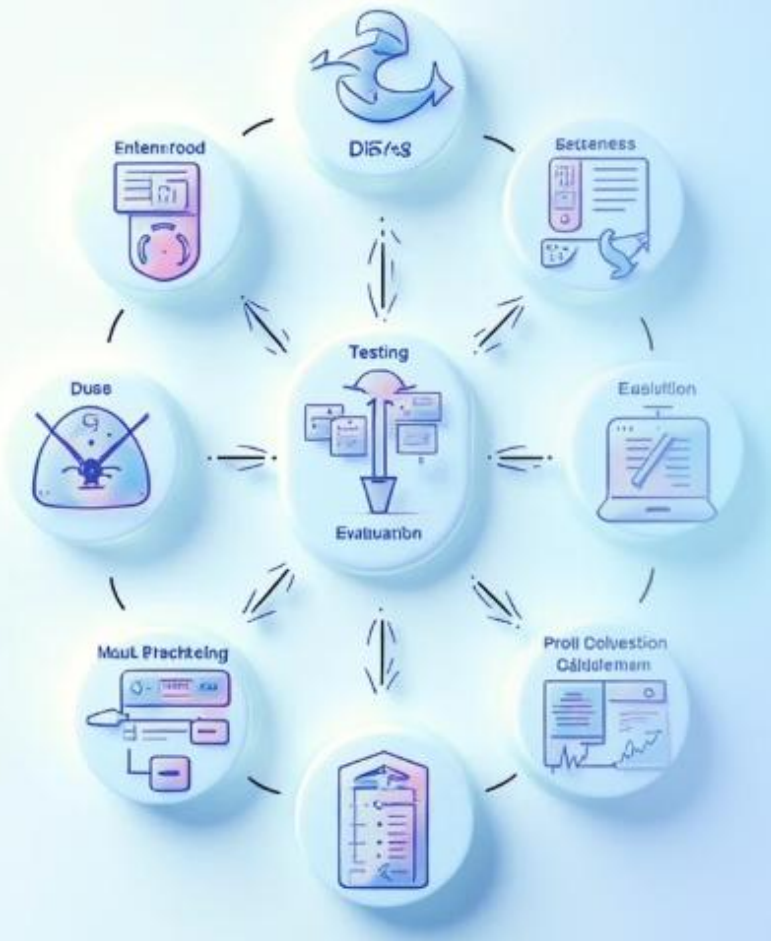
Gauss–Jordan Method

reliable and widely used in numerical computation.

4

Structured Programming Techniques

functions, modular design, loops, conditions.



6. Structured Development Process

The development followed a systematic and professional approach.

Problem Definition

To create a program that:

1	2	3
Accepts an augmented matrix of size (n × (n+1)).	Performs Gauss–Jordan elimination.	Outputs the final solution in clear numerical form.

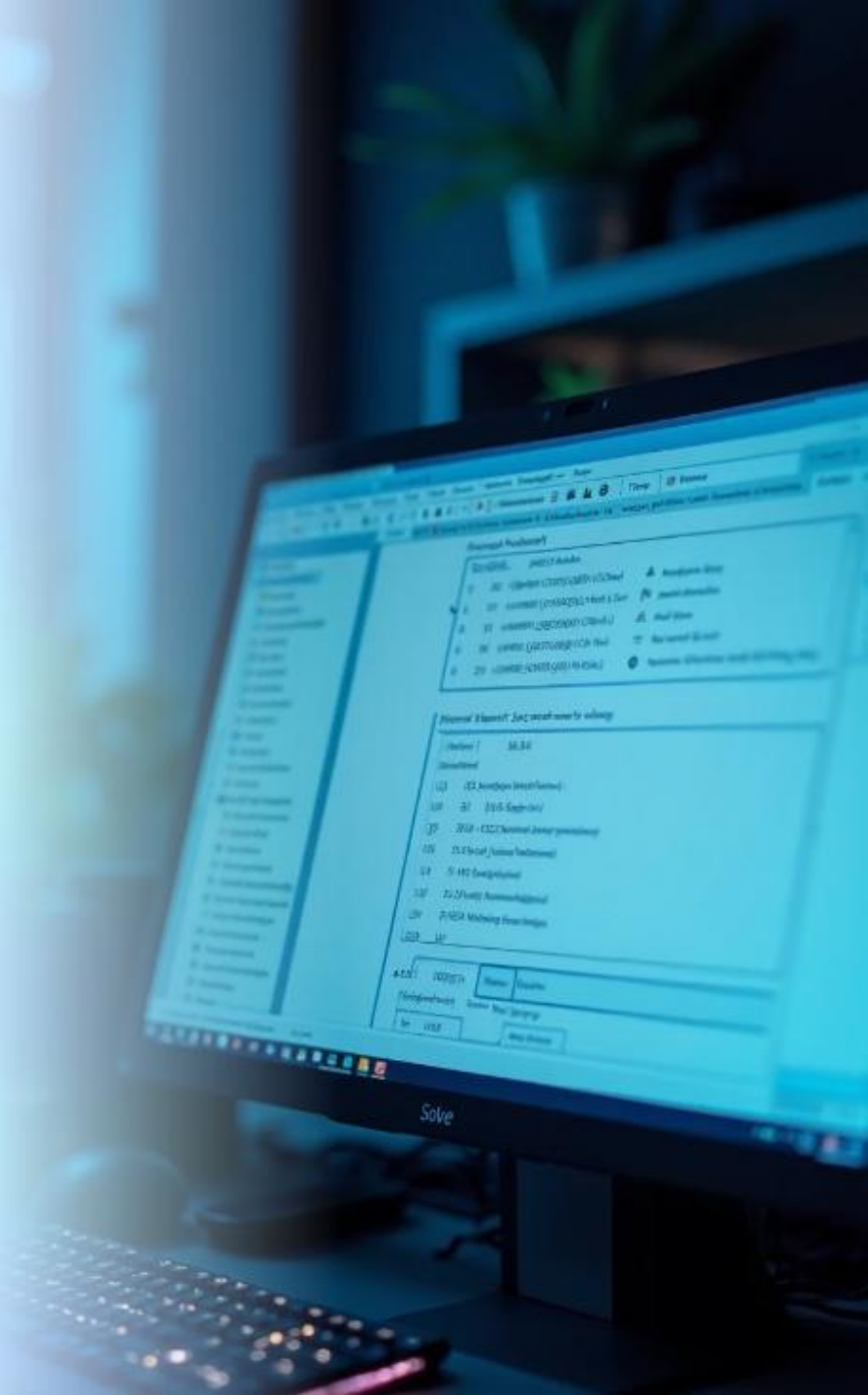
Requirement Analysis

Functional Requirements

- User inputs number of equations.
- User inputs all matrix values.
- Program validates input.
- Program performs row operations.
- Program outputs solution.

Non-Functional Requirements

- Accuracy of computation.
- Efficiency (handled using NumPy).
- Proper error handling.
- Clear and formatted output.



Top-Down Design / Modularization

The program is divided into smaller modules:

1 Main Program

Calls functions in correct order.

2 Module 1

get_matrix(): Input rows, compute columns, input matrix values, typecasting and validation.

3 Module 2

print_matrix(): Print matrix neatly.

4 Module 3

gauss_jordan(): Perform elimination, handle pivot = 0, normalize rows, eliminate other rows.

5 Module 4

print_solution(): Display values of unknowns.

This modular design improves readability, maintainability, and debugging.

Algorithm Development (Gauss–Jordan Method)

Step 1

Input number of unknowns.

Step 2

Input augmented matrix values.

Step 3

For each pivot row:

- If pivot is zero → swap with a lower row.
- Divide the entire row by pivot to make it 1.
- For all other rows, make pivot column = 0.

Step 4

After matrix becomes RREF, last column gives solutions.

Step 5

Print variable values:

- x1 = A[0][-1].
- x2 = A[1][-1], etc.

Testing and Refinement

The program was tested with multiple input scenarios:

1

Test 1

Normal solvable system - Works correctly; gives expected solutions.

2

Test 2

Pivot zero case - Program swaps rows automatically.

3

Test 3

Invalid numeric input - Program catches errors using try-except.

4

Test 4

Floating-point numbers - Outputs correct results.

5

Test 5

Large values - Works well due to NumPy handling.

Refinements made:

1

Improved error messages.

2

Added row swapping logic.

3

Ensured pivot normalization.

7. Conclusion

This project successfully implements a robust Gauss–Jordan Linear Equation Solver. Through this project, key programming principles were applied:

- Input validation
- Typecasting
- Conditional logic
- Nested loops
- Functions and modular design
- Use of Python libraries
- Implementation of a mathematical algorithm

The final program is efficient, modular, and user-friendly. It demonstrates both theoretical understanding and practical programming skills, meeting all academic and functional requirements.

Thank You