Project Report

Simplified Matrix Calculator

Submitted by:

Mohit Agrawal
Registration No: 25BCE10672

Course:
CSE1021

Faculty:
Sandeep Monga sir

Slot:
A14+D11+D12

Contents

1 Introduction

The **Simplified Matrix Calculator** is a computer program built using Python. It is designed to help students and engineers solve linear algebra problems quickly and accurately.

In mathematics, working with matrices (grids of numbers) can be difficult. Operations like finding the "Inverse" or multiplying two large matrices together involve a lot of calculation steps. If you make even one small mistake, the final answer is wrong. This project solves that problem by using the computer to do the math instantly. It is a simple tool that runs in the command line, allowing users to verify their homework or perform complex calculations without needing expensive software.

2 Problem Statement

The Issue

Engineering students often spend hours solving matrix problems by hand. While it is important to learn the method, doing the calculation manually is prone to human error. A simple slip in addition can ruin the result of a 3×3 determinant.

The Need

There is a lack of simple, lightweight tools for this. Most scientific calculators are hard to use for large matrices, and software like MATLAB is too heavy for simple tasks. Students need a quick, text-based tool where they can type in their numbers and get the answer immediately.

3 Functional Requirements

These are the specific features the system provides:

1. **Matrix Creation:** The system allows the user to define the size (rows and columns) of a matrix and input the numbers.
2. **Single Matrix Operations:**
   - **Determinant:** Calculates the determinant if the matrix is square.
   - **Inverse:** Calculates the inverse if the matrix is non-singular.
   - **Transpose:** Flips the rows and columns.
3. **Combined Operations:**
   - **Addition:** Adds two matrices ($A + B$) if they are the same size.
   - **Subtraction:** Subtracts Matrix B from A.
   - **Multiplication:** Multiplies $A \times B$ after checking if the dimensions match.
4. **Comparison:** Checks if two matrices are identical.

4 Non-functional Requirements

- **Usability:** The program must be easy to read. It uses a clear step-by-step menu that asks simple "Yes/No" questions (1 or 0).
- **Reliability:** The program should not crash. If a user types a letter instead of a number, the system catches the error and asks them to try again.
- **Accuracy:** It uses the `NumPy` library, which is an industry standard for scientific computing, ensuring decimal precision.
- **Efficiency:** The calculations happen instantly, even for larger matrices, because of optimized code.

5 System Architecture

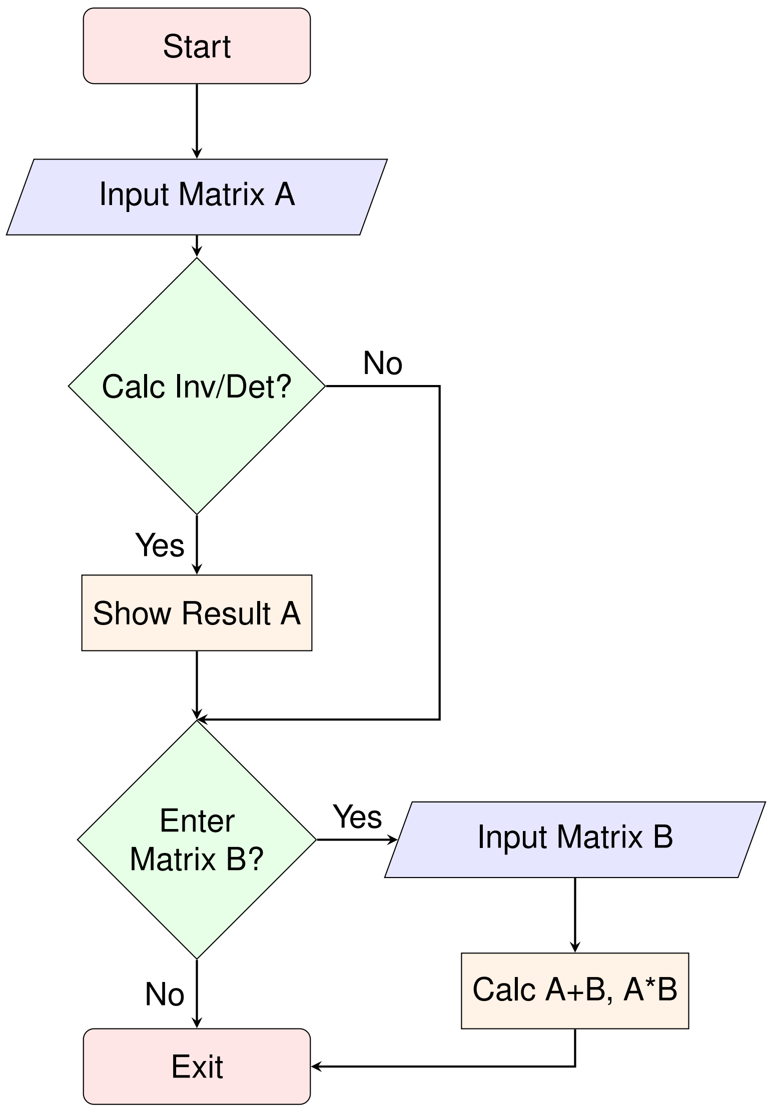
The project follows a simple **Procedural Architecture**. This means the code runs in a straight line, step-by-step.

- **Input Layer:** This part of the code asks the user for rows, columns, and values. It handles the "User Interface" in the terminal.
- **Logic Layer:** This checks the rules. For example, it checks *"Is the number of columns in A equal to the number of rows in B?"* before allowing multiplication.
- **Computation Layer:** This is where the `NumPy` library does the heavy lifting to calculate the answers.

6 Design Diagrams

Workflow Diagram

This flowchart shows how a user interacts with the system.



Use Case Diagram Description

- **Actor:** User (Student/Engineer)
- **Use Cases:**
  - Define Matrix Size
  - Input Data
  - Request Inverse/Determinant
  - Request Combined Operations (Add/Mult)
  - View Results

7 Design Decisions & Rationale

1. **Choice of Language (Python):** We chose Python because it is very easy to read and write. It handles lists and numbers better than languages like C++ or Java for this type of quick scripting.

2. **Using NumPy Library:** Writing code to multiply matrices from scratch requires three nested loops, which is slow and hard to debug. `NumPy` is a pre-built library that is optimized for speed. Using it makes the code cleaner and reliable.
3. **Single-Line Input:** Initially, we asked users to input numbers one by one. This was annoying for a 3×3 matrix (9 inputs). We decided to let users type all numbers in one line separated by spaces to make it faster.

8 Implementation Details

The project is built in a single Python file. Here are the key technical details:

- **Libraries:** The code imports numpy as np.
- **Input Handling:** We use input().split() to take the whole row of numbers at once, map them to floats, and then put them into a list.
- **Reshaping:** We use np.array(list).reshape(rows, cols) to turn that simple list into a mathematical matrix grid.
- **Safety Loops:** We use while True loops for inputs. If the user makes a mistake, the loop runs again until they enter valid data.

9 Screenshots / Results

Since this is a text-based report, below is a simulation of the output screen when the program is running.

```
--- 1. Set Up Matrix A ---
Enter the number of rows for Matrix A: 2
Enter the number of columns for Matrix A: 2
Enter 4 entries for Matrix A: 1 2 3 4

Matrix A has been created:
[[1. 2.]
 [3. 4.]]

--- 2. Operations on Matrix A ---
Calculate Inverse of A? (1 for Yes / 0 for No): 1
The Inverse of Matrix A is:
[[-2.   1. ]
 [ 1.5 -0.5]]

Do you want to enter Matrix B? (1/0): 1
... [User enters Matrix B as 2 2 2 2] ...
The sum of Matrix A and B is:
[[3. 4.]
 [5. 6.]]
```

Terminal Output Simulation

10 Testing Approach

We used "Manual Black-Box Testing". This means we ran the program and tried different inputs to see if it behaved correctly.

- **Test Case 1 (Standard):** We entered a standard Identity matrix. *Result:* The determinant was calculated as 1.0 (Correct).
- **Test Case 2 (Error Input):** We entered text "abc" instead of rows. *Result:* The program printed "Invalid input" and asked again (Pass).
- **Test Case 3 (Math Logic):** We tried to add a 2×2 matrix to a 3×3 matrix. *Result:* The program showed an error saying dimensions must match (Pass).

11 Challenges Faced

- **Understanding Dimensions:** It was tricky to write the logic for matrix multiplication. We had to ensure that the Columns of A matched the Rows of B, otherwise the program would crash.
- **Handling Inverses:** If a matrix has a determinant of 0, it cannot be inverted. Initially, our program crashed on these matrices. We had to add a check if determinant != 0 to fix this.

12 Learnings & Key Takeaways

Through this project, we learned:

1. How to use external libraries like `NumPy` to simplify complex coding tasks.
2. The importance of "Input Validation" never trust that the user will type the right thing.
3. How to structure a program logically so that the code flows from setup to operations to results.

13 Future Enhancements

In the future, this project can be improved by:

- Adding a **Graphical User Interface (GUI)** so users can click buttons instead of typing.
- Allowing users to **save results** to a text file.
- Adding support for solving linear equations (like finding x, y, z).

14 References

- NumPy Official Documentation: https://numpy.org/doc/
- Python 3.10 Documentation: https://docs.python.org/3/
- Course Syllabus: Linear Algebra and Python Programming.