Assignment 3

Arithmetic Operation on Matrices - 10 points

Pedram Pasandide

Due Date: 2025, 25 July

Introduction

Matrix arithmetic operations are fundamental in various fields, including computer graphics, physics simulations, machine learning, and numerical analysis. In this assignment, we explore the implementation of basic matrix arithmetic operations in C, where matrices are randomly initialized and manipulated through a command-line interface. The goal is to understand how matrix operations work programmatically and how to handle memory efficiently in C.

Matrix operations are widely used in engineering, physics, and data science. They play a critical role in solving systems of linear equations, transforming geometric data, and performing complex computations in artificial intelligence. By implementing these operations from scratch, students will gain a deeper understanding of memory management, pointer arithmetic, and command-line interactions in C.

In this assignment, students will implement a C program to perform the following matrix operations:

- Addition: $A + B$
- Subtraction: $A - B$
- Multiplication: $A \times B$
- Transposition: A^T
- Solving systems of linear equations: $Ax = B$

A system of linear equations consists of multiple linear equations involving the same set of variables. In general, a system of n linear equations with n unknowns can be written in the form:

$$Ax = B$$

where:

- A is an $n \times n$ matrix (known coefficients),
- x is an $n \times 1$ column vector (unknowns to be solved),
- B is an $n \times 1$ column vector (known constants).

The objective is to find x that satisfies the given system. Please read Appendix for more information about solving $Ax = B$. In this assignment, you are free to use any method that works best for you. Consider efficiency, numerical stability, and ease of implementation when choosing your approach.

The program will accept user inputs via the command line, dynamically allocate memory for matrices, and ensure proper deallocation. Error handling will be incorporated to manage incorrect input sizes and operations. Usually input matrices are given by users or computed during the execution. However, here for simplicity, we initialize input matrices A and B randomly to test the program.

Programming Logic (8 points)

The program will be executed using the following command format:

```
./myAO <print_flag> <operation> <other inputs>
```

where:

- `print_flag`: If `0`, no output is printed; if `1`, input matrices **and** results are displayed. Usually the results of arithmetic operations on matrices are used within the program and there is no need to print. However, we'll need to print the matrices in this assignment to test the program.
- `operation`: Specifies the arithmetic operation including `+`, `-`, `*`, `T` (stands for **T**ranspose), or `s` (stands for **S**olve).
- `other inputs`: Dimensions of matrices `A` and `B`.

Examples of command execution:

- Addition $(A + B)$ (already implemented):

```
./myAO 1 + nA mA nB mB
```

With `1` all matrices including `A`, `B`, and the result of `A+B` will be printed in the terminal. `nA`, `mA`, `nB`, and `mB` are integer values showing the matrix dimensions. The matrix A has the size of $n_A \times m_A$ (`nA` number of rows and `mA` number of columns). The dimension of matrix B is $n_B \times m_B$.

```
./myAO 0 + nA mA nB mB
```

With the above command line, the same operation $A + B$ is conducted, however, nothing will be printed.

- Subtraction $(A - B)$ (0.5/8 points):

```
./myAO 0 - nA mA nB mB
```

- Multiplication $(A \times B)$ (1/8 points):

```
./myAO 0 \* nA mA nB mB
```

- Transposition (A^T) (1/8 points):

```
./myAO 1 T nA mA
```

- Solving $(Ax = B)$ (3/8 points):

```
./myAO 0 s nA mA nB mB
```

The program must check the criteria for all above operations. If the criteria is not satisfied, an error message is displayed.

Code Organization (1/8 points)

The program consists of the following files:

- `test.c`: Handles command-line input, initializes matrices, and calls operations.

- `myAO.c`: Implements arithmetic operations such as addition, subtraction, multiplication, linear solver, and transposition. Here, every function that is a part of your library is implemented.
- `myAO.h`: Header file declaring arithmetic operation functions.
- `utility.c`: Implements matrix initialization and printing functions. Only helper functions are implemented here. For example, a printing function does not have anything to do with arithmetic operation on matrices.
- `utility.h`: Header file declaring utility functions.

Students will develop the matrix arithmetic functions in `myAO.c` by following provided comments and specifications. Matrices `A` and `B` are randomly initialized with `fillRandom()` function to facilitate testing.

Error Handling and Memory Management (2/8 points)

The program must handle all possible errors, including:

- Dimension mismatches in operations (e.g., matrix addition and multiplication rules).
- Invalid command-line inputs.
- Memory allocation failures.

Before exiting due to an error, the program must deallocate any dynamically allocated memory to prevent memory leaks.

Report and Makefile (1.5 points)

Create a PDF file named `Report.pdf` that includes the documentation for your program. The report should cover essential information, such as how to compile and run the program. **Create a section called Appendix in your report and include all the source code files as text (not screenshots)**. A report file without following the above structure will **not** be accepted.

Additionally, write a `Makefile` to compile your program. The `Makefile` should define the rules for compiling. Include a short description about your `Makefile` in your report, describing the `Makefile` including every flag being used and all the rules.

Submission On Avenue to Learn

Submit your assignment as:

1. `test.c`,
2. `myAO.c`,
3. `myAO.h`,
4. `utility.c`,
5. `utility.h`,
6. `Makefile`,
7. and a report PDF file **named** `Report.pdf`.

Please do **NOT** submit any zip files.

Appendix: Solving Systems of Linear Equations

Solving systems of linear equations is fundamental in various fields, including:

- Physics (e.g., electrical circuit analysis, mechanics),
- Engineering (e.g., structural analysis, control systems),
- Computer Science (e.g., graphics transformations, machine learning),
- Economics (e.g., optimization problems, input-output models),
- Data Science (e.g., regression analysis, numerical methods).

There are multiple ways to solve the linear system $Ax = B$, depending on properties of A (such as invertibility and sparsity). Some common methods include:

- **Direct methods:**
  - Gaussian elimination
  - LU decomposition

  - Cholesky decomposition (for symmetric positive definite matrices)

- **Iterative methods (for large sparse systems):**
  - Jacobi method
  - Gauss-Seidel method
  - Conjugate Gradient method (for symmetric positive definite matrices)

- **Matrix inverse (when A is invertible, though not always numerically efficient):**

$$x = A^{-1}B$$

Many programming languages provide built-in functions for solving linear systems, such as:

- `numpy.linalg.solve` (Python)
- `Eigen::MatrixXd::colPivHouseholderQr().solve()` (C++)
- `Matlab's A \B` operator

Please test the result of your implementations with one of the above built-in methods to ensure the results are correct.