Matrix Manipulator

A C++ Operator Overloading Adventure

The project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Science

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Anvitha.K

AP22111260014

Under the Guidance of

Subhankar Ghatak

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[November 2023]

Table of contents

ABSTRACT

This report presents a C++ program for matrix manipulation using object-oriented programming concepts. The program defines a Matrix class that allows users to perform various matrix operations, including addition, subtraction, scalar multiplication, matrix multiplication, matrix transposition, matrix inversion, and element-wise division. Users can interact with the program through a menu-driven interface, making it user-friendly and accessible.

The Matrix class implements matrix operations efficiently and handles errors gracefully, providing informative error messages to the user when necessary. Additionally, the program includes support for square matrices and the calculation of matrix inverses.

This report details the design and implementation of the program, covering the structure of the Matrix class and its member functions. It also explains the Gauss-Jordan elimination method used to calculate matrix inverses. Users are guided through the program's features and functionalities.

The provided C++ program offers a versatile and convenient tool for performing common matrix operations and can be a valuable asset for various mathematical and scientific applications. The code is well-organized, modular, and readable, making it suitable for educational purposes and practical use.

INTRODUCTION

This report presents a C++ program designed to facilitate matrix manipulation and analysis using object-oriented programming (OOP) principles. The program offers a user-friendly interface for working with matrices, offering functionalities such as matrix addition, subtraction, scalar multiplication, matrix multiplication, matrix transposition, and element-wise division.

It also provides the means to calculate the inverse of square matrices, a fundamental operation in solving systems of linear equations and other mathematical problems. The Matrix class, the core of the program, encapsulates matrix data and operations within an OOP framework, promoting reusability and code clarity.

The report provides a comprehensive overview of the program's design, implementation, and features, explaining fundamental data structures, algorithms, and error-handling mechanisms. It guides users through the menu-driven interface, demonstrating how to apply various matrix operations.

This program serves as a valuable tool for matrix manipulation and analysis in mathematical research, scientific computing, and educational settings. Its well-organized, modular structure allows users to explore the world of matrices intuitively and efficiently.

BACKGROUND

Matrices are fundamental in various fields, including mathematics, physics, computer science, engineering, statistics, machine learning, economics, signal processing, operations research, biology and genetics, social sciences, finance, environmental science, and control systems.

They are used in linear algebra, quantum mechanics, computer graphics, engineering, statistics, machine learning, and data science to represent and manipulate data and relationships. In mathematics, they are used for solving complex mathematical problems, while in physics, they are used for describing the behavior of subatomic particles.

In computer science, they are used for structural analysis, statistics, machine learning, and data science to reduce the dimensionality of data. In economics, they are used for input-output models, signal processing, linear programming, and genetic data analysis.

In social sciences, they are used for social network analysis, portfolio analysis, environmental modeling, and state-space representations in control systems. Matrices are a foundational tool in modern science and technology, enabling researchers, scientists, engineers, and data analysts to model, analyze, and solve complex problems.

PROBLEM STATEMENT

You are tasked with designing a matrix calculator program that performs various operations on matrices. The program should allow users to input two matrices, and perform operations such as addition, subtraction, scalar multiplication, matrix multiplication, transpose, matrix inverse, and element-wise division. The program should provide a user-friendly menu for the operations.

Matrix operations menu:

1. Addition
2. Subtraction
3. Scalar Multiplication
4. Matrix Multiplication
5. Transpose A
6. Transpose B
7. Inverse of A
8. Inverse of B
9. Division of A
10. Division of B
11. Exit

IMPLEMENTATION DETAILS

Matrix Class:

The program defines a Matrix class to represent matrices. It includes a private member variable data for storing matrix elements, as well as rows and cols to specify the number of rows and columns in the matrix.

- A constructor is provided to initialize the matrix with a given number of rows and columns. The matrix is initially filled with zeros.

Matrix Operations:

The program implements several matrix operations within the Matrix class, including Matrix addition (operator+) Matrix subtraction (operator-) Scalar multiplication (operator*) Matrix multiplication (operator*) Matrix division (elementwise) (operator/) Matrix negation (operator-) Matrix transpose (transpose function) Matrix inverse (inverse function) using Gauss-Jordan elimination.

Input and Output:

- The program takes user input to specify the dimensions of matrices and to input the elements of matrices A and B.
- It provides a menu for selecting various matrix operations and displays the results.
- Users can choose from operations such as addition, subtraction, multiplication, division, and more.

CLASS DIAGRAM

```
-----------------------------------
|          <<Matrix>>           |
-----------------------------------
| - data: std::vector<std::vector<double>> |
| - rows: int                    |
| - cols: int                   |
-----------------------------------
| + Matrix(rows: int, cols: int)   |
| + inverse(): Matrix             |
| + operator+(other: Matrix): Matrix |
| + operator-(other: Matrix): Matrix |
| + operator*(scalar: double): Matrix |
| + operator*(other: Matrix): Matrix  |
| + operator/(scalar: double): Matrix  |
| + operator-(): Matrix           |
| + transpose(): Matrix          |
| + print()                     |
| + setElement(row: int, col: int, value: double) |
-----------------------------------


       |
       |
       v
---------------------------------------------
|                Main                 |
---------------------------------------------
| + main()                            |
---------------------------------------------
```

```
      |
      |   uses
      v
-----------------------------------
|          Console Input          |
-----------------------------------
| + cin                         |
| + cout                         |
-----------------------------------
```

CODE

```
#include <iostream>
#include <vector>
#include <stdexcept>

class Matrix {
private:
   std::vector<std::vector<double>> data;
   int rows, cols;

public:
   Matrix(int rows, int cols) : rows(rows), cols(cols) {
      data.resize(rows, std::vector<double>(cols, 0.0));
   }

   // ... other matrix operations (e.g., addition, subtraction, multiplication) ...

   // Function to calculate the matrix inverse using Gauss-Jordan elimination
   Matrix inverse() const {
      if (rows != cols) {
         throw std::runtime_error("Matrix must be square to find the inverse.");
      }

      // Create an augmented matrix [A | I] where 'I' is the identity matrix
      Matrix augmented(rows, 2 * cols);

      // Initialize the augmented matrix with 'A' on the left and 'I' on the right
      for (int i = 0; i < rows; i++) {
         for (int j = 0; j < cols; j++) {
                augmented.data[i][j] = data[i][j];
                if (i == j) {
                    augmented.data[i][j + cols] = 1.0;
                }
            }
        }

        // Apply Gauss-Jordan elimination to transform the left side into the identity
    matrix
        // The result will be [I | A^-1]
        for (int i = 0; i < rows; i++) {
            // Find the pivot element
            double pivot = augmented.data[i][i];
            if (pivot == 0.0) {
                throw std::runtime_error("Matrix is not invertible.");
            }

            // Scale the row to make the pivot element 1
            for (int j = 0; j < 2 * cols; j++) {
                augmented.data[i][j] /= pivot;
            }

            // Eliminate other rows to make elements above and below the pivot zero
            for (int k = 0; k < rows; k++) {
                if (k != i) {
                    double factor = augmented.data[k][i];
                    for (int j = 0; j < 2 * cols; j++) {
                        augmented.data[k][j] -= factor * augmented.data[i][j];
                    }
                }
            }
        }

        // Extract the inverse from the right side of the augmented matrix
        Matrix result(rows, cols);
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result.data[i][j] = augmented.data[i][j + cols];
            }
        }

        return result;
    }

    // ... other member functions (e.g., printing the matrix, setting elements) ...

    // Overload the + operator for matrix addition
    Matrix operator+(const Matrix& other) const {
        if (rows != other.rows || cols != other.cols) {
            throw std::runtime_error("Matrix dimensions must match for addition.");
        }

        Matrix result(rows, cols);
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result.data[i][j] = data[i][j] + other.data[i][j];
            }
        }
        return result;
    }
```

```
    // Overload the - operator for matrix subtraction
    Matrix operator-(const Matrix& other) const {
         if (rows != other.rows || cols != other.cols) {
            throw std::runtime_error("Matrix dimensions must match for subtraction.");
        }

        Matrix result(rows, cols);
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result.data[i][j] = data[i][j] - other.data[i][j];
            }
        }
        return result;
    }

    // Overload the * operator for scalar multiplication
    Matrix operator*(double scalar) const {
        Matrix result(rows, cols);
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result.data[i][j] = data[i][j] * scalar;
            }
        }
        return result;
    }

    // Overload the * operator for matrix multiplication
    Matrix operator*(const Matrix& other) const {
        if (cols != other.rows) {
            throw std::runtime_error("Number of columns in the first matrix must match
the number of rows in the second matrix.");
```

```
    }

    Matrix result(rows, other.cols);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < other.cols; j++) {
            for (int k = 0; k < cols; k++) {
                result.data[i][j] += data[i][k] * other.data[k][j];
            }
        }
    }
    return result;
}

// Overload the / operator for matrix division (element-wise)
Matrix operator/(double scalar) const {
    if (scalar == 0) {
        throw std::runtime_error("Division by zero is not allowed.");
    }

    Matrix result(rows, cols);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result.data[i][j] = data[i][j] / scalar;
        }
    }
    return result;
}

// Overload the unary - operator for matrix negation
Matrix operator-() const {
    Matrix result(rows, cols);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result.data[i][j] = -data[i][j];
        }
    }
    return result;
}

// Transpose the matrix
Matrix transpose() const {
    Matrix result(cols, rows);
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            result.data[j][i] = data[i][j];
        }
    }
    return result;
}

// Print the matrix
void print() const {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            std::cout << data[i][j] << "\t";
        }
        std::cout << std::endl;
    }
}
void setElement(int row, int col, double value) {
        if (row >= 0 && row < rows && col >= 0 && col < cols) {
            data[row][col] = value;
        } else {
            throw std::runtime_error("Invalid row or column index.");
        }
    }
};

int main() {
    int rows, cols;
    std::cout << "Enter the number of rows for the matrices: ";
    std::cin >> rows;
    std::cout << "Enter the number of columns for the matrices: ";
    std::cin >> cols;

    Matrix A(rows, cols);
    Matrix B(rows, cols);
    Matrix C(rows, cols); // Define the result matrix

    std::cout << "Enter the elements of matrix A:" << std::endl;
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            double element;
            std::cout << "A[" << i << "][" << j << "]: ";
            std::cin >> element;
            A.setElement(i, j, element);
        }
    }

    std::cout << "Enter the elements of matrix B:" << std::endl;
```

```
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        double element;
        std::cout << "B[" << i << "][" << j << "]: ";
        std::cin >> element;
        B.setElement(i, j, element);
    }
}

int choice;

do {
    std::cout << "\nMatrix Operations Menu:\n";
    std::cout << "1. Addition\n";
    std::cout << "2. Subtraction\n";
    std::cout << "3. Scalar Multiplication\n";
    std::cout << "4. Matrix Multiplication\n";
    std::cout << "5. Transpose A\n";
    std::cout << "6. Transpose B\n";
    std::cout << "7. Inverse of A\n";
    std::cout << "8. Inverse of B\n";
    std::cout << "9. Division of A\n";
    std::cout << "10. Division of B\n";
    std::cout << "11. Exit\n";
    std::cout << "Enter your choice: ";
    std::cin >> choice;

    switch (choice) {
        case 1:
            C = A + B;
    std::cout << "Result of Addition:\n";
    C.print();
    break;
case 2:
    C = A - B;
    std::cout << "Result of Subtraction:\n";
    C.print();
    break;
case 3:
    double scalar;
    std::cout << "Enter the scalar value: ";
    std::cin >> scalar;
    C = A * scalar;
    std::cout << "Result of Scalar Multiplication:\n";
    C.print();
    break;
case 4:
    C = A * B;
    std::cout << "Result of Matrix Multiplication:\n";
    C.print();
    break;
case 5:
    C = A.transpose();
    std::cout << "Transpose of A:\n";
    C.print();
    break;
case 6:
    C = B.transpose();
    std::cout << "Transpose of B:\n";
    C.print();
    break;
case 7:
    // Calculate the inverse of A
    try {
        C = A.inverse();
        std::cout << "Inverse of A:\n";
        C.print();
    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
    }
    break;
case 8:
    // Calculate the inverse of B
    try {
        C = B.inverse();
        std::cout << "Inverse of B:\n";
        C.print();
    } catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << std::endl;
    }
    break;
case 9:
    double divisorA;
    std::cout << "Enter the divisor for A: ";
    std::cin >> divisorA;
    C = A / divisorA;
    std::cout << "Result of Division for A:\n";
    C.print();
    break;
case 10:
```

```
                double divisorB;
                std::cout << "Enter the divisor for B: ";
                std::cin >> divisorB;
                C = B / divisorB;
                std::cout << "Result of Division for B:\n";
                C.print();
                break;
            case 11:
                std::cout << "Exiting the program.\n";
                break;
            default:
                std::cout << "Invalid choice. Please enter a valid option.\n";
        }
    } while (choice != 11);

    return 0;
}
```

Private Members:

1. data: 2D vector to store matrix elements.
2. rows: Number of rows in the matrix.
3. cols: Number of columns in the matrix.

Public Members:

1. Constructor: Initializes the matrix with the specified number of rows and columns.
2. inverse(): Calculates and returns the inverse of the matrix.
3. Operator Overloads: +, -, * (scalar and matrix multiplication), /, - (unary), ==.
4. transpose(): Transposes the matrix.
5. print(): Prints the matrix elements.
6. setElement(): Sets the value of a matrix element at a specific position.

Usage:

1. Run the program.
2. Enter the number of rows and columns for matrices A and B.
3. Choose an operation from the menu and follow any additional prompts.

https://onlinegdb.com/nCslUfREH

ERROR HANDLING

The C++ program provides error-handling mechanisms to ensure appropriate responses during matrix operations. These mechanisms include matrix dimension validation, division by zero checks, matrix inverse validation, matrix inversion check, invalid index check, and error reporting.

Matrix dimension validation checks the dimensions of matrices when performing addition and subtraction operations. If dimensions don't match, a std::runtime error is thrown. Division by zero check prevents division by zero in the operator/function. Inverse validation checks if the matrix is square, and if not, a std::runtime error is thrown. Inversion check checks if the matrix is invertible during the Gauss-Jordan elimination process. If the matrix is not invertible, a std::runtime error is thrown.

Invalid index check ensures the provided row and column indices are valid before attempting to set an element. If the row or column index is out of bounds, a std::runtime error is thrown. Error reporting uses the std::runtime error exception class to provide informative error messages, which are displayed to the user through std::cerr (standard error) to differentiate them from regular output.

CONCLUSION

The provided C++ program defines a Matrix class and demonstrates various matrix operations, including addition, subtraction, multiplication, division, and more. It also allows the user to perform these operations interactively through a menu-driven console application.

This program also serves as a practical example of object-oriented programming in C++ and demonstrates the use of classes and operator overloading for matrix manipulations. It also incorporates exception handling to ensure robustness and user-friendly interaction. Depending on your needs, you can further extend the program to support additional matrix operations or enhance its user interface.

REFERENCES

OpenAI Documentation, "GPT-3.5 Architecture Overview," URL: https://www.openai.com/documentation/gpt-3.5.

https://www.geeksforgeeks.org/operator-overloading-cpp/

https://www.mygreatlearning.com/blog/operator-overloading-in-cpp/

Special thanks to Dr subhankar Ghatak, for guidance on the operator overloading concept.

Thank you