Report: Matrix Operations in C

Hardik Gera, McMaster University

4ex

1 Introduction

In this programming assignment, the objective is to create a C program capable of performing scientific operations on matrices. The program generates two random 2D arrays, A and B, with specified dimensions, and supports operations such as addition, subtraction, multiplication, and solving linear systems of equations. The input format requires specifying matrix dimensions and the desired operation, with an optional parameter for printing the matrices and results. The implementation involves three main files: math matrix.c, functions.h, and functions.c, with functions like addMatrices() and solveLinearSystem() handling the various operations. Testing includes verifying correctness and identifying potential issues like segmentation faults, with debugging tools used to pinpoint errors

2 Linear Equation Algorithm

```
1  This function solves a system of linear equations represented by the ↩
       matrix equation Ax = b using Gaussian elimination with back ↩
       substitution. Here, A is a square matrix of size N1 x N1, x is the ↩
       unknown solution vector of size N1 x 1, and b is the constant vector ↩
       of size N1 x 1.
2
3  Algorithm:
4
5  1)Dimension Check:
6
7  The function verifies that the dimensions of matrix A (N1 x M1) are ↩
       compatible for solving a linear system. It checks if:
8  N1 (number of rows) is equal to M1 (number of columns) to ensure a square ↩
       matrix.
9  M2 (number of columns in B) is equal to 1 (single solution vector).
10 If the dimensions are incompatible, an error message is printed, and the ↩
       program exits.
11
12 2)Create Augmented Matrix:
13
14 A new matrix aug of size N1 x (M1 + 1) is created. This augmented matrix ↩
       combines the coefficient matrix A and the constant vector b.
15 For each row i of the augmented matrix:
```

```
16 Columns from 0 to M1-1 are copied from the corresponding row i of matrix A.
17 The last column (M1) of the augmented matrix is filled with the elements from the corresponding row i of vector b.
18
19 3)Gaussian Elimination (Forward Elimination):
20
21 This step transforms the augmented matrix into an upper triangular matrix.
22 For each row i (from 0 to N1-2):
23 Iterate through rows j below the current row (j = i+1 to N1-1).
24 Calculate a factor r (aug[i][i] divided by aug[j][i]).
25 For each column k (from 0 to M1):
26 Subtract the product of r and aug[j][k] from the corresponding element aug[i][k] in the current row.
27 This process eliminates the elements below the diagonal in each column, resulting in an upper triangular matrix.
28
29 4)Back Substitution:
30
31 This step solves for the unknown variables in the solution vector x.
32 Since the augmented matrix is now upper triangular, we can solve for each element of x starting from the last row (N1-1) and working backwards.
33 For each row i (from N1-1 down to 0):
34 Initialize a sum (sum) to 0.
35 Iterate through rows j above the current row (j = i+1 to N1-1).
36 Add the product of aug[i][j] and the already solved element x[j][0] to the sum.
37 Calculate the value of x[i][0] using the formula: x[i][0] = (aug[i][M1] - sum) / aug[i][i]
38 This formula considers the constant term on the right side (aug[i][M1]) and subtracts the contribution of already solved variables (sum) to isolate the current variable (x[i][0]).
39
40 5)Solution Vector:
41
42 After back substitution, the solution vector x contains the solutions to the system of linear equations represented by Ax = b. Each element x[i][0] corresponds to the solution for the i-th variable.
43 \section{Code Functionality}
```

3 Rest of the code Explanation

```
1 generateMatrix(int rows, int cols, double matrix[rows][cols]): This function generates a random matrix with specified dimensions and fills
```

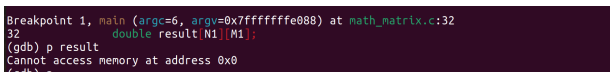

Figure 1: cannot access memory

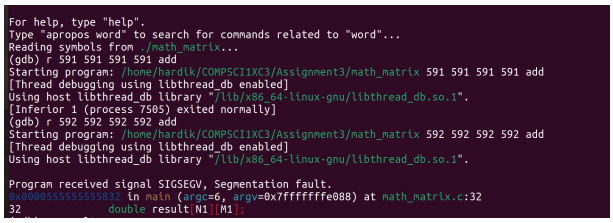

Figure 2: segmentation fault on line 32

it with random values between -10 and 10.

2 printMatrix(int rows, int cols, double matrix[rows][cols]): This function prints a matrix to the console. It iterates through each element of the matrix and prints its value with two decimal places.

3 addMatrices(int N1, int M1, double A[N1][M1], int N2, int M2, double B[N2][M2], double result[N1][M1]): This function adds two matrices of compatible dimensions. It checks if the dimensions of the matrices are compatible for addition, and then performs element-wise addition and stores the result in the result matrix.

4 subtractMatrices(int N1, int M1, double A[N1][M1], int N2, int M2, double B[N2][M2], double result[N1][M1]): Similar to the addition function, this function subtracts one matrix from another. It checks for compatible dimensions and performs element-wise subtraction.

5 multiplyMatrices(int N1, int M1, double A[N1][M1], int N2, int M2, double B[N2][M2], double result[N1][M2]): This function multiplies two matrices with compatible dimensions. It checks if the number of columns in the first matrix equals the number of rows in the second matrix, and then performs matrix multiplication and stores the result in the result matrix.

---

4 Segmentation Fault

When taking N1, M1, N2, M2 as 592, I encountered 'Segmentation Fault on line 32'. While going through the execution line by line using gdb I found the line which was causing the segmentation fault. Here, the address allocated to result is '0x0' which indicates that the program is attempting

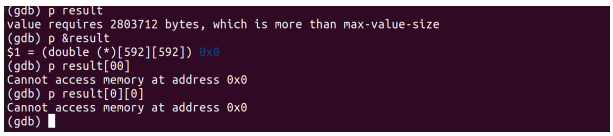

Figure 3: Cannot Access memory

to allocate result to a NULL address and hence the memory allocation has failed. One possible reason for this can be that the size of the result array is exponentially large and it cannot be allocated an address in the stack which is just 8-9 Megabytes.

4.1 Makefile

```
1
2  CC = gcc
3  CFLAGS = -Wall -Wextra -std=c99 -g
4  EXECUTABLE = math_matrix
5  SRC = math_matrix.c
6  OBJSRC = functions.c
7  OBJ = functions.o
8  HLP = functions.h
9
10 $(EXECUTABLE): $(SRC) $(OBJ)
11     $(CC) $(CFLAGS) -o $(EXECUTABLE) $(SRC) $(OBJ)
12 $(OBJ): $(OBJSRC) $(HLP)
13     $(CC) -c $(OBJSRC) $(CFLAGS) -o $(OBJ)
14 clean:
15     rm -f math_matrix functions.o
```

5 Code

The complete C code for the Matrix Operations is provided on Avenue to Learn. Here's a text form code of the file.

5.1 Main

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int main(int argc, char *argv[])
6  {
7      // Seed the random number generator
8      srand(time(NULL));
9
10     // Check for valid number of arguments
11     if (argc < 6 || argc > 7)
12     {
```

```
13          printf("Usage: %s nrows_A ncols_A nrows_B ncols_B <operation> [↩
                print]\n", argv[0]);
14          return 1;
15      }
16
17      // Get matrix dimensions from arguments
18      int N1 = atoi(argv[1]);
19      int M1 = atoi(argv[2]);
20      int N2 = atoi(argv[3]);
21      int M2 = atoi(argv[4]);
22
23      // Generate random matrices A and B
24      double A[N1][M1], B[N2][M2];
25      generateMatrix(N1, M1, A);
26      generateMatrix(N2, M2, B);
27
28      // Determine operation based on argument
29      char *operation = argv[5];
30      if (strcmp(operation, "add") == 0)
31      {
32          double result[N1][M1];
33          addMatrices(N1, M1, A, N2, M2, B, result);
34
35          // Print matrices A, B, and result (if requested)
36          if (argc == 7 && strcmp(argv[6], "print") == 0)
37          {
38              printf("Matrix A:\n");
39              printMatrix(N1, M1, A);
40              printf("\n"); // New line after Matrix A
41              printf("Matrix B:\n");
42              printMatrix(N2, M2, B);
43              printf("\n"); // New line after Matrix B
44              printf("Result of Addition:\n");
45              printMatrix(N1, M1, result);
46          }
47      }
48      else if (strcmp(operation, "subtract") == 0)
49      {
50          double result[N1][M1];
51          subtractMatrices(N1, M1, A, N2, M2, B, result);
52
53          // Print matrices A, B, and result (if requested)
54          if (argc == 7 && strcmp(argv[6], "print") == 0)
55          {
56              printf("Matrix A:\n");
57              printMatrix(N1, M1, A);
58              printf("\n"); // New line after Matrix A
```

```
59              printf("Matrix B:\n");
60              printMatrix(N2, M2, B);
61              printf("\n"); // New line after Matrix B
62              printf("Result of Subtraction:\n");
63              printMatrix(N1, M1, result);
64          }
65      }
66      else if (strcmp(operation, "multiply") == 0)
67      {
68          double result[N1][M2]; // Result will have dimensions N1 x M2
69          multiplyMatrices(N1, M1, A, N2, M2, B, result);
70
71          // Print matrices A, B, and result (if requested)
72          if (argc == 7 && strcmp(argv[6], "print") == 0)
73          {
74              printf("Matrix A:\n");
75              printMatrix(N1, M1, A);
76              printf("\n"); // New line after Matrix A
77              printf("Matrix B:\n");
78              printMatrix(N2, M2, B);
79              printf("\n"); // New line after Matrix B
80              printf("Result of Multiplication:\n");
81              printMatrix(N1, M2, result); // Print with adjusted dimensions
82          }
83      }
84      else if (strcmp(operation, "solve") == 0)
85      {
86          double x[N1][M2]; // Assuming solution vector x has same ↩
                dimensions as B
87          solveLinearSystem(N1, M1, A, N2, M2, B, x);
88
89          // Print matrices A, B, and solution vector x (if requested)
90          if (argc == 7 && strcmp(argv[6], "print") == 0)
91          {
92              printf("Matrix A:\n");
93              printMatrix(N1, M1, A);
94              printf("\n"); // New line after Matrix A
95              printf("Matrix B:\n");
96              printMatrix(N2, M2, B);
97              printf("\n"); // New line after Matrix B
98              printf("Result of x=B/A:\n");
99              printMatrix(N1, M2, x); // Print with adjusted dimensions
100         }
101     }
102     else
103     {
104         printf("Invalid operation: %s\n", operation);
```

```
105         return 1;
106     }
107 
108     return 0;
109 }
```

5.2 Function.c

```
1  // CODE: include necessary library(s)
2  
3  #include "functions.h"
4  #include <stdio.h>
5  #include <stdlib.h> // For rand() and srand()
6  #include <time.h>   // For time()
7  #include <math.h>
8  
9  // Function to generate a random matrix with values between -10 and 10
10 void generateMatrix(int rows, int cols, double matrix[rows][cols])
11 {
12     for (int i = 0; i < rows; i++)
13     {
14         for (int j = 0; j < cols; j++)
15         {
16             matrix[i][j] = (double)rand() / (double)(RAND_MAX) * 20 - 10; ↩
                   // Generate random number and adjust to range
17         }
18     }
19 }
20 
21 // Function to print a matrix
22 void printMatrix(int rows, int cols, double matrix[rows][cols])
23 {
24     for (int i = 0; i < rows; i++)
25     {
26         for (int j = 0; j < cols; j++)
27         {
28             printf("%f ", matrix[i][j]); // Print with 2 decimal places
29         }
30         printf("\n");
31     }
32 }
33 
34 // Function to add two matrices with compatible dimensions
35 void addMatrices(int N1, int M1, double A[N1][M1], int N2, int M2, double ↩
```

```
    B[N2][M2], double result[N1][M1])
{
    if (N1 != N2 || M1 != M2)
    {
        printf("Error: Matrices have incompatible dimensions for addition.\n");
        exit(1);
    }

    for (int i = 0; i < N1; i++)
    {
        for (int j = 0; j < M1; j++)
        {
            result[i][j] = A[i][j] + B[i][j]; // Add corresponding elements
        }
    }
}

// Function to subtract two matrices with compatible dimensions
void subtractMatrices(int N1, int M1, double A[N1][M1], int N2, int M2, double B[N2][M2], double result[N1][M1])
{
    if (N1 != N2 || M1 != M2)
    {
        printf("Error: Matrices have incompatible dimensions for subtraction.\n");
        exit(1);
    }

    for (int i = 0; i < N1; i++)
    {
        for (int j = 0; j < M1; j++)
        {
            result[i][j] = A[i][j] - B[i][j]; // Subtract corresponding elements
        }
    }
}

// Function to multiply two matrices with compatible dimensions (A x B)
void multiplyMatrices(int N1, int M1, double A[N1][M1], int N2, int M2, double B[N2][M2], double result[N1][M2])
{
    if (M1 != N2)
    {
        printf("Error: Matrices have incompatible dimensions for
```

```
            multiplication.\n");
        exit(1);
    }

    for (int i = 0; i < N1; i++)
    {
        for (int j = 0; j < M2; j++)
        {
            result[i][j] = 0;
            for (int k = 0; k < M1; k++)
            {
                result[i][j] += A[i][k] * B[k][j]; // Perform element-wise
                    multiplication and summation
            }
        }
    }
}
// Function to solve a linear system of equations (Ax = b) using Gaussian
    elimination with back substitution

void solveLinearSystem(int N1, int M1, double A[N1][M1], int N2, int M2,
    double B[N2][M2], double x[N1][M2])
{
    if (N1 != N2 || M2 != 1)
    {
        printf("Error: Matrices have incompatible dimensions for solving
            the linear system.\n");
        exit(1);
    }
    double aug[N1][M1 + 1];

    // Create augmented matrix
    for (int i = 0; i < N1; i++)
    {
        for (int j = 0; j < M1 + 1; j++)
        {
            if (j == M1)
            {
                aug[i][j] = B[i][0];
            }
            else
            {
                aug[i][j] = A[i][j];
            }
        }
    }

```

```
    // Perform Gaussian elimination (forward elimination)
    for (int i = 0; i < N1 - 1; i++)
    {
        for (int j = i + 1; j < N1; j++)
        {
            double r = aug[i][i] / aug[j][i];
            for (int k = 0; k < M1 + 1; k++)
            {
                aug[j][k] = aug[i][k] - r * aug[j][k];
            }
        }
    }

    // Perform back substitution
    x[N1 - 1][0] = aug[N1 - 1][M1] / aug[N1 - 1][N1 - 1];
    for (int i = N1 - 2; i >= 0; i--)
    {
        double sum = 0;
        for (int j = i + 1; j < N1; j++)
        {
            sum += aug[i][j] * x[j][0];
        }
        x[i][0] = (aug[i][M1] - sum) / aug[i][i];
    }
}
```

6 Sources

1. https://github.com/ckoreli/universal-ppm-image-filter
2. https://www.numerade.com/ask/question/write-a-c-program-which-a-take-number-of-column-and-rows-of-two-matrices-from-userb-take-elements-of-two-matrices-93122/
3. https://chat.openai.com/(Used for taking reference of functions and report.)