

Assignment 3 report

Yisi Liu

March 2024

1 Introduction

Linear algebra, matrices, and the ability to work with them through 2D arrays in C are very useful skills, which was practiced in this assignment. In this report, I will explain Gaussian Elimination, which I used as the algorithm for solving linear systems of equations in the form of $Ax = B$ where A and B are given. I will also explain why my code crashes with a segmentation fault when typing “./math_matrix.c 512 512 512 512 add” in the terminal. Finally, my complete codes are attached in Appendix 1.

2 The Algorithm

$Ax = B$ is one way to represent a linear system of equations, and one very code-friendly, algorithmic way to solve it is by taking the column vector B , augmenting A with it, and row-reducing the resulting matrix. The answer will be the final column.

To row-reduce, first, divide the first row by the first number in the row to get a leading 1 in the first column of the first row. Then, add the appropriate multiple of the first row to all subsequent rows so that everything in the first column below the leading 1 sum to a zero. This appropriate multiple will be the negative of the first entry of the row in question.

Next, repeat the same steps except with the second entry of the second number.

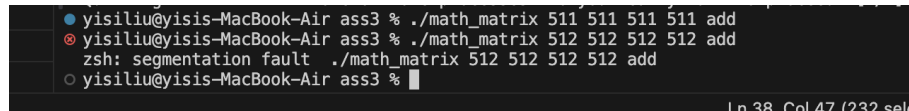
Repeat until the final row is reached. This will reach the second-last column because A must be square to ensure a unique solution, and B is a column vector.

The so-called backwards phase then begins, where appropriate multiples of rows are added to the rows above them to turn everything in a column with a leading 1, other than said leading 1, into a zero. The appropriate multiple is, again, the negative of the entry in the same column as the specific column being cleared for whichever row being added to at any given time.

Once the augmented matrix resembles an identity matrix with a random column vector stuck to its right side, that last column will be x .

3 Segmentation fault

A segmentation fault occurs when attempting to run “./math_matrix.c 512 512 512 512 add” in terminal, but not right before.

A terminal window on a MacBook Air showing a sequence of commands and their outputs. The first command is './math_matrix 511 511 511 511 add', which runs successfully. The second command is './math_matrix 512 512 512 512 add', which results in a 'zsh: segmentation fault ./math_matrix 512 512 512 512 add' error. The prompt then returns to the user. The terminal background is dark, and the text is light-colored. There are some status icons on the left side of the terminal window.

```
• yisiliu@yisis-MacBook-Air ass3 % ./math_matrix 511 511 511 511 add
⊗ yisiliu@yisis-MacBook-Air ass3 % ./math_matrix 512 512 512 512 add
zsh: segmentation fault ./math_matrix 512 512 512 512 add
○ yisiliu@yisis-MacBook-Air ass3 %
```

Any number bigger than 512 also produces the same error in my computer.

This is because this segmentation fault is caused by stack overflow, just like the example shown in lecture of trying to create an array

“int largeArray[2098100]” within the main() function because what that does is that it’s asking for memory. It asks to be allocated the space in memory to store an integer array of size 2098100, and since it is a local variable because it’s within the main() function, it’s allocated memory in the stack where information about function calls, including local variables, are stored, automatically. The stack has a set, restricted size, however, and exceeding that size (a stack overflow) counts as accessing illegal memory, which is a segmentation fault.

Within my code, I first automatically allocate on stack “double A[512][512]”, then “double B[512][512]”, then simultaneously “resultadd[512][512]” and “resultx[512][512]” both doubles. The sum of the space in memory I’m asking for just exceeds (’cause 511 is still within limits) my system specifications (someone else’s system specifications may allow up to 591, or anything else) for the stack size and I produce a segmentation fault due to stack overflow.

This is clearly evident when stepping through my code with lldb, and seeing that the second I hit the line with resultadd and resultx, the error appears because that is the line where everything together finally exceeds the limit of my stack.

```

C functions.c | C functions.h | C math_matrix.c x | Makefile
C math_matrix.c > main(int, char * [])
9  int main(int argc, char *argv[]) {
28  // CODE: Generate random matrices A with size N1*M1 and B with size N2*M2
29
30  double A[N1][M1];
31  generateMatrix(N1, M1, A);
32  if (print) printMatrix(N1, M1, A);
33
34  double B[N2][M2];
35  generateMatrix(N2, M2, B);
36  if (print) printMatrix(N2, M2, B);
37
38  double resultadd[N1][M1], resultx[N1][M2];
39  // CODE: call the functions as needed
40  if (strcmp(argv[5], "add") == 0)
41  {
42      addMatrices(N1, M1, A, N2, M2, B, resultadd);
43      if (print) printMatrix(N1, M1, resultadd);
44  }
45  }
46
47  PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
48
49  Process 12870 stopped
50  * thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 3.1
51  frame #0: 0x0000000100003a80 math_matrix`main(argc=6, argv=0x000000016fdff250) at math_matrix.c:38:22
52  35  generateMatrix(N2, M2, B);
53  36  if (print) printMatrix(N2, M2, B);
54  37
55  -> 38  double resultadd[N1][M1], resultx[N1][M2];
56  39  // CODE: call the functions as needed
57  40  if (strcmp(argv[5], "add") == 0)
58  41  {
59  Target 0: (math_matrix) stopped.
60  (lldb) s
61  Process 12870 stopped
62  * thread #1, queue = 'com.apple.main-thread', stop reason = EXC_BAD_ACCESS (code=2, address=0x16f603ff8)
63  frame #0: 0x0000000182ed4e7c libsystem_pthread.dylib`__chkstk_darwin + 60
64  libsystem_pthread.dylib:
65  -> 0x182ed4e7c <+0>: ldur x11, [x11, #-0x8]
66  0x182ed4e80 <+64>: mov x10, sp
67  0x182ed4e84 <+68>: cmp x9, #0x1, lsl #12 ; =0x1000
68  0x182ed4e88 <+72>: b.lo 0x182ed4ea0 ; <+96>
69  Target 0: (math_matrix) stopped.
70  (lldb)
71
72  Ln 38, Col 47 (232 selected) Spaces: 4 UTF-8 LF {} C N

```

4 Appendix 1

The following are my complete code files.

```

1  // CODE: include necessary library(s)
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include "functions.h"
5  #include "functions.c"
6  #include <string.h>
7  #include <time.h>
8
9  int main(int argc, char *argv[]) {
10     srand(time(NULL));
11
12     // Check if the number of arguments is correct
13     if (argc < 6 || argc > 7) {
14         printf("Usage: %s nrows_A ncols_A nrows_B ncols_B <
15         operation> [print]\n", argv[0]);
16         return 1;
17     }
18 }

```

```

18     int print = 0;
19     if (argc == 7)
20     {
21         print = 1;
22     }
23
24     int N1 = atoi(argv[1]);
25     int M1 = atoi(argv[2]);
26     int N2 = atoi(argv[3]);
27     int M2 = atoi(argv[4]);
28     // CODE: Generate random matrices A with size N1*M1 and B with
    size N2*M2
29
30     double A[N1][M1];
31     generateMatrix(N1, M1, A);
32     if (print) printMatrix(N1, M1, A);
33
34     double B[N2][M2];
35     generateMatrix(N2, M2, B);
36     if (print) printMatrix(N2, M2, B);
37
38     double resultadd[N1][M1], resultx[N1][M2];
39     // CODE: call the functions as needed
40     if (strcmp(argv[5], "add") == 0)
41     {
42         addMatrices(N1, M1, A, N2, M2, B, resultadd);
43         if (print) printMatrix(N1, M1, resultadd);
44     }
45     if (strcmp(argv[5], "subtract") == 0)
46     {
47         subtractMatrices(N1, M1, A, N2, M2, B, resultadd);
48         if (print) printMatrix(N1, M1, resultadd);
49     }
50
51     if (strcmp(argv[5], "multiply") == 0)
52     {
53         multiplyMatrices(N1, M1, A, N2, M2, B, resultx);
54         if (print) printMatrix(N1, M2, resultx);
55     }
56     if (strcmp(argv[5], "solve") == 0)
57     {
58         solveLinearSystem(N1, M1, A, N2, M2, B, resultx);
59         if (print) printMatrix(N1, M2, resultx);
60     }
61
62     return 0;
63 }

```

Listing 1: math_matrix.c

```

1 // CODE: include necessary library(s)
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include "functions.h"
5
6 // Function to generate a random matrix
7 void generateMatrix(int rows, int cols, double matrix[rows][cols])
    {

```

```

8 // CODE: Generate random numbers between -10 and 10
9
10 for (int i = 0; i < rows; i++)
11 {
12     for (int j = 0; j < cols; j++)
13     {
14         matrix[i][j] = (rand()%20000000)/(double)1000000 - 10;
15     }
16 }
17
18
19 // Function to print a matrix
20 void printMatrix(int rows, int cols, double matrix[rows][cols]) {
21     // CODE: to print the matrix
22
23     for (int i = 0; i < rows; i++)
24     {
25         for (int j = 0; j < cols; j++)
26         {
27             printf("%lf ", matrix[i][j]);
28         }
29         printf("\n");
30     }
31     printf("\n");
32 }
33
34 // Function to add two matrices
35 void addMatrices(int N1, int M1, double A[N1][M1], int N2, int M2,
36                 double B[N2][M2], double result[N1][M1])
37 {
38     // CODE: check for the condition
39     if (N1 != N2 || M1 != M2)
40     {
41         printf("Matrix sizes incompatible for addition. Result
42         matrix will be random numbers or a system default.\n");
43         return;
44     }
45
46     for (int i = 0; i < N1; i++)
47     {
48         for (int j = 0; j < M1; j++)
49         {
50             result[i][j] = A[i][j] + B[i][j];
51         }
52     }
53 }
54
55 void subtractMatrices(int N1, int M1, double A[N1][M1], int N2, int
56                      M2, double B[N2][M2], double result[N1][M1])
57 {
58     // CODE: check for the condition
59     if (N1 != N2 || M1 != M2)
60     {
61         printf("Matrix sizes incompatible for subtraction. Result
62         matrix will be random numbers or a system default.\n");
63         return;
64     }
65
66     for (int i = 0; i < N1; i++)

```

```

61     {
62         for (int j = 0; j < M1; j++)
63         {
64             result[i][j] = A[i][j] - B[i][j];
65         }
66     }
67 }
68
69 void multiplyMatrices(int N1, int M1, double A[N1][M1], int N2, int
    M2, double B[N2][M2], double result[N1][M2])
70 {
71     if (M1 != N2)
72     {
73         printf("Matrix sizes incompatible for multiplication.
    Result matrix will be random numbers or a system default.\n");
74         return;
75     }
76
77     for (int i = 0; i < N1; i++)
78     {
79         for (int j = 0; j < M2; j++)
80         {
81             double sum = 0;
82             for (int k = 0; k < M1; k++)
83             {
84                 sum += (A[i][k] * B[k][j]);
85             }
86             result[i][j] = sum;
87         }
88     }
89 }
90
91 //creating forward-declared helper functions for row operations
    seperately to keep organized:
92 void interchange(int rows, int cols, double matrix[rows][cols], int
    idx1, int idx2);
93
94 void multiply(int rows, int cols, double matrix[rows][cols], int
    idx, double constant);
95
96 void addrow(int rows, int cols, double matrix[rows][cols], int idx1
    , int idx2, double constant);
97
98 void solveLinearSystem(int N1, int M1, double A[N1][M1], int N2,
    int M2, double B[N2][M2], double x[N1][M2])
99 {
100     if (N1 != N2 || M2 != 1 || M1 != N1) //if A isn't square then
        there is no unique solution
101     {
102         printf("Matrix sizes incompatible for obtaining a solution
        for a linear system. Result matrix will be random numbers or a
        system default.\n");
103         return;
104     }
105
106     double aug[N1][M1+1]; //augmenting A with B
107

```

```

108     for (int i = 0; i < N1; i++)
109     {
110         for (int j = 0; j < M1; j++)
111         {
112             aug[i][j] = A[i][j];
113         }
114     }
115     for (int i = 0; i < N2; i++)
116     {
117         aug[i][M1] = B[i][0]; //actual index of the last extra row
118                                //is M1 + 1 - 1
119     }
120
121     for (int i = 0; i < N1; i++) //essentially iterating down the
122                                //diagonal of A since it's square to perform Gaussian Elimination
123                                //on the augmented matrix where B tags along
124     {
125         multiply(N1, (M1+1), aug, i, (1/aug[i][i]));
126         for (int j = i+1; j < M1; j++)
127         {
128             addrow(N1, (M1+1), aug, i, j, (-aug[j][i]));
129         }
130     }
131
132     for (int i = (N1 - 1); i >= 0; i--) //now for the backwards
133                                         //phase
134     {
135         for (int j = i-1; j >= 0; j--)
136         {
137             addrow(N1, (M1+1), aug, i, j, (-aug[j][i]));
138         }
139     }
140
141     for (int i = 0; i < N2; i++) //the last column that tagged along
142                                     //is now the answer
143     {
144         x[i][0] = aug[i][M1];
145     }
146 }
147
148 void interchange(int rows, int cols, double matrix[rows][cols], int
149                 idx1, int idx2)
150 {
151     double spare[cols];
152
153     for (int j = 0; j < cols; j++)
154     {
155         spare[j] = matrix[idx2][j];
156     }
157
158     for (int j = 0; j < cols; j++)
159     {
160         matrix[idx2][j] = matrix[idx1][j];
161     }
162
163     for (int j = 0; j < cols; j++)

```

```

159     {
160         matrix[idx2][j] = spare[j];
161     }
162 }
163
164 void multiply(int rows, int cols, double matrix[rows][cols], int
165             idx, double constant)
166 {
167     for (int j = 0; j < cols; j++)
168     {
169         matrix[idx][j] = (matrix[idx][j])*constant;
170     }
171 }
172 void addrow(int rows, int cols, double matrix[rows][cols], int idx1
173            , int idx2, double constant)
174 {
175     for (int j = 0; j < cols; j++)
176     {
177         matrix[idx2][j] += (matrix[idx1][j])*constant;
178     }
179 }

```

Listing 2: functions.c