SPL-1 Project Report

Different Matrix Operations and Multiplicative Update with Hadamard Initialization

Submitted by

Ahnaf Mubashshir Mobin

BSSE Roll No. : 1232

BSSE Session: 2019-2020

Submitted to

Dr. Mohammad Shoyaib

Professor, IIT, DU

Institute of Information Technology

University of Dhaka

29-05-2021

Table of Contents

Index of Figures

1. Introduction

Non-negative matrix factorization (NMF) is a popular technique allowing the decomposition of two-dimensional non-negative data as a linear combination of meaningful elements. Given a non-negative matrix V, we find non-negative matrix factors W and H such that:

$$W \approx VH \tag{1.1}$$

NMF can be applied to the statistical analysis of multivariate data in the following manner. Given a set of multivariate n-dimensional data vectors, the vectors are placed in the columns of an n x m matrix V where m is the number of examples in the data set. This matrix is then approximately factorized into an n x r matrix W and an r x m matrix H. Usually r is chosen to be smaller than nor m, so that Wand H are smaller than the original matrix V. This results in a compressed version of the original data matrix.

What is the significance of the approximation in Eq. (1.1)? It can be rewritten column by column as V ~ WH, where v and h are the corresponding columns of V and H. In other words, each data vector v is approximated by a linear combination of the columns of W, weighted by the components of h. Therefore, W can be regarded as containing a basis that is optimized for the linear approximation of the data in V. Since relatively few basis vectors are used to represent many data vectors, good approximation can only be achieved if the basis vectors discover structure that is latent in the data.

2. Background of the Project

There are multiple rules to factorize a matrix. One of them is **Multiplicative Update.** Here we split a matrix into to matrices. The dimension of the two matrices can be higher or lower. For this rule to be executed, I needed some more rules like **Euclidean Distance**, Finding **Standard Deviation**, **Hadamard Matrix** creation, picking value **Normal Distribution.**

2.1) Standard Deviation: We use **SD** to normalize a matrix. There are many techniques to normalize a matrix. One of them is dividing every element of a matrix by its SD. Normalizing a matrix means making its values lower, so that we can find the factorized matrix easily. Otherwise, it will give us higher error. The formula which is used to find **SD** is:

$$\boldsymbol{\sigma} = \sqrt{\frac{\sum(x_i - \mu)^2}{N}} \tag{2.1}$$

Here,

μ= average of the inputted matrix.

x_i= values of the inputted matrix.

N= total elements of the matrix.

σ= standard deviation.

2.2) Hadamard Matrix: A Hadamard matrix is an $n \times n$ matrix with elements ± 1 and mutually orthogonal columns. For example:

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \quad \textbf{(2.2)}$$

The entries here are either **+1** or **-1**. And the dimension of this will always be $\mathbf{2^n x 2^n}$, where n=**{N}** & N means set of all natural numbers**.** Here is the formula to create a **Hadamard Matrix:**

$$\sqrt{\frac{N}{K}} DH_nS \quad \textbf{(2.3)}$$

2.3) Normal distribution: Sometimes I needed to initialize something (matrix/values) using random number. So, I used the **Normal Distribution** to do this. Like, for a matrix $M_{10000x10000}$ I could not give user input; so, picked values from the normal distribution for this randomly to see is my multiplicative update working or not. So, for making this run I used this formula:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \quad \textbf{(2.4)}$$

x= a random number.

μ= average of the inputted matrix.

σ= standard deviation.

$f(x)$= probability density function/ value picked from a normal distribution.

2.4) Euclidean Distance: When a matrix is factorized into two matrices & after that if we multiply those two matrices, we need to get the exact same matrix. But the problem that arises here, we can't get the same old matrix back. There remains some error. We minimize the error and try to get a nearly same old matrix. This error is minimized after every iteration in multiplicative update. And after every iteration, we calculate the error. We multiply the factorized matrices & compare it with the old matrix and thus we find the error. For finding this error I used **Euclidian Distance.** This means, the distance between two values. As, I was comparing two same shell of a matrix, so, Iused **Euclidean Distance**. The formula for **Euclidian Distance** is:

$$d(p.q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \quad \textbf{(2.5)}$$

2.5) Multiplicative Update: It is actually a non-negative matrix factorization process, where a matrix is factorized into two reduced dimension matrices. Let, **V** be a matrix which will be factorized into two matrices **W** & **H**. So, **V=WH**. Now, in multiplicative update, in the first iteration we keep **W** or **V** one them constant and change another one. Let it be **W**. So, in the first iteration, we are keeping **W** constant & changing **V**. Then, again in the second iteration we keep **V** constant & change **W**. The formula:

$$H \leftarrow H + \eta_H . (W^T X - W^T W) \qquad W \leftarrow W + \eta_w . (X H^T - W H H^T)$$

(2.6)

$$H_{a\mu} \leftarrow H_{a\mu} \frac{(W^T V)_{a\mu}}{(W^T W H)_{a\mu}} \qquad W_{ia} \leftarrow W_{ia} \frac{(V H^T)_{ia}}{(W H H^T)_{ia}}$$

2.6) Gauss-Seidel: The Gauss-Seidel method is a technique for solving the equations of the linear equation solving one at a time in sequence, and uses previously computed results as soon as they are available.

$$x_i^{(k)} = \frac{b_i - \sum_{j>i} a_{i\,j} x_j^{(k)} - \sum_{j<i} a_{i\,j} x_j^{(k-1)}}{a_{ii}} \qquad \textbf{(2.7)}$$

3. Description of the Project

Matrix factorization is process to factorize a matrix into two matrices. For doing so, I needed some functions like **SD to normalize matrix, Initialization using Hadamard Matrix, Random Number generator using Normal Distribution, Euclidean Distance to find error, Multiplicative Update to Factorize a Matrix & Gauss-Seidel to Solve Linear Equation.**

3.1) SD to Normalize Matrix: We need to normalize a matrix before it is factorized. Because if we normalize the matrix, we can find the factorized matrices easily. So, we use **SD** to normalize the matrices.

3.2) Initialization using Hadamard Matrix: We need to initialize the matrices, which is allocated for store the factorized matrices. We need to initialize in a way that, the factorization process gets easier. So, we use here, Hadamard matrix initialization.

3.3) Random Number generator using Normal Distribution: Sometimes, we need to take billion or trillion inputs. We can't input them manually. So, we need a random number generator. This must be made in such and efficient way that the factorization process gets easier. So, we use here Normal distribution. We pick the values from normal distribution and we input it in the matrix.

3.4) Multiplicative Update to Factorize Matrix: This is the process by which we factorize the inputted matrix. When we initialized two matrices, made to store the factorized matrices. Then, we need to make it nearly accurate. For that, we keep one constant and change another. Then, in the next iteration, we keep constant the other one & change the first one. These two process happens multiple times, and after several times happened, we get the factorized matrices.

3.5) Euclidean Distance to find error: When we are in the stage multiplicative update and we are continuing the iterations, how would it know that when to stop. Here comes error finding. We find the error using Euclidean Distance. When the error isless than e particular epsilon we stop the iterations and get the answer.

3.6) Gauss-seidel to solve linear equations: Gauss-seidel is a linear equation solving process that is done using the help of matrix. We take the inputs of the left-hand side coefficients as matrix and also the right-hand side values as matrix. And then we solve it using the Gauss-Seidel process.

4. Implementation and Testing

The main goal of my project is to factorize a matrix into two pieces. For doing such a thing, I needed to implement many other rules and I needed to implement them in a sequence. At first, I needed to take input of a matrix which will be factorized. I took the input in to way **User input** & **Random input from PC**. Then I normalized the inputted matrix using **SD.** So that, I could factorize it faster. After this is done, I allocated memory for two matrices to store the values of factorized matrices and initialized them using **Hadamard Matrix**. Then I used **Multiplicative Update** for factorizing the matrices and at every iteration I needed to find error, to check should I stop now or continue. So, I used **Euclidean distance** to check the error, After, several iterations, I reached my goal and thus I got the factorized matrices.

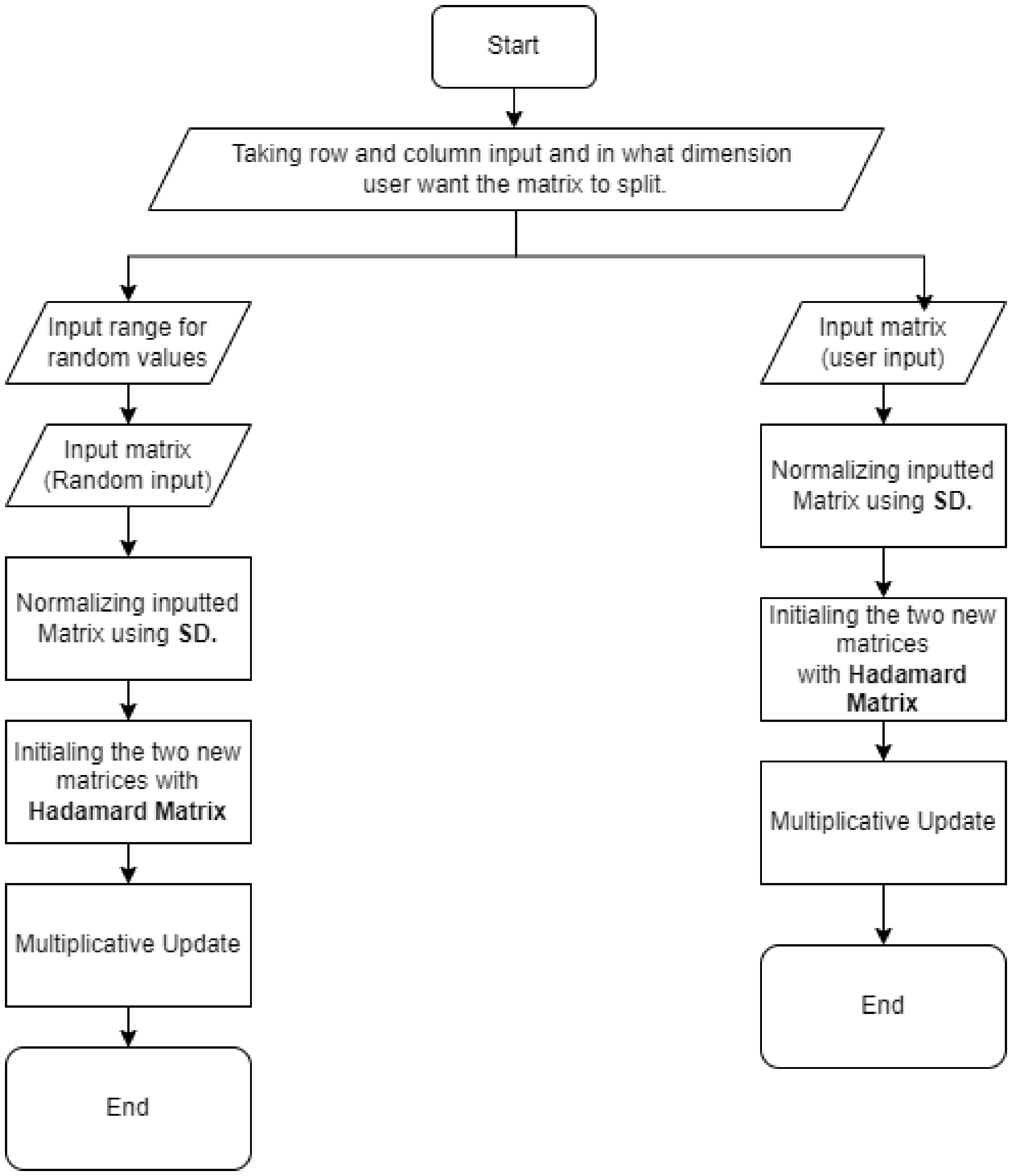

Start
Taking row and column input and in what dimension user want the matrix to split.
Input range for random values
Input matrix (Random input)
Normalizing inputted Matrix using SD.
Initialing the two new matrices with Hadamard Matrix
Multiplicative Update
End
Input matrix (user input)
Normalizing inputted Matrix using SD.
Initialing the two new matrices with Hadamard Matrix
Multiplicative Update
End

Figure 4.1: Flow chart of Matrix factorization

4.1) Normalizing Matrix using SD: To normalize a matrix, I, at first, took a matrix as input (it can be random or user input). Then, I calculated the **SD** of the matrix and at last, I divided all the elements by the **SD**. Thus, my inputted matrix is normalized.

4.2) Initializing Matrices using Hadamard Initialization: As, I needed to initialize the factorized matrices with something before starting multiplicative update; so, I used Hadamard matrix to initialize those empty matrices. I could also use random numbers but This helped me to find the factorized matrices a lot faster.

4.3) Normal Distribution to generate Random Numbers: For matrix of dimension $\boldsymbol{M_{10000x10000}}$ it's impossible to give a user input here. So, I inputted values here by picking values from **Normal distribution.** For doing so, I used the built-in random number generator. And used the formula of **Normal Distribution** step by step.

4.4) Euclidian Distance to Calculate error: Here error means,

Beside this factorization, I added some features. These are the actions of matrix. I implemented **Matrix Addition, Matrix Subtraction, Matrix Multiplication, Finding Inverse of a Matrix, Finding Determinant of a Matrix, Transpose Matrix & Solving Linear equations using Matrix.**

4.5) Matrix Addition: To add Two matrices, I used elementwise addition process. Here I took input of row and columns and then took input of two matrices and then added them element wise.

4.6) Matrix Subtraction: To subtract two matrices, I used elementwise Subtraction process. Here I took input of row and columns and then took input of two matrices and then subtracted them element wise.

4.7) Matrix Multiplication: To multiply two matrices, I used dot multiplication process. At first, I took how much matrices a user want to multiply. Then I took input of row and columns and then took input of two matrices and then multiplied them. Every row of the first is multiplied with every column of the second matrix. After finishing a multiplication, I took another matrix & its' row/column as input and also took the choice that the matrix he/she want to multiply result matrix is in the position before or after the result matrix. And then again multiply them. It continues until all the matrix are multiplied.

4.8) Inverse of a Matrix: For finding **Inverse of a Matrix,** I used **Pseudo Inverse.** Here I we at first transpose the inputted matrix and then I multiplied it with the main matrix. After that, I did the inverse using normal inverse operation where I found the determinant at first and check if the determinant is 0 or not. If not, then I did the inverse finding adjoint and then dividing it elementwise by its’ determinant. I used **Pseudo Inverse** because it can find inverse of any matrix (square or non-square).

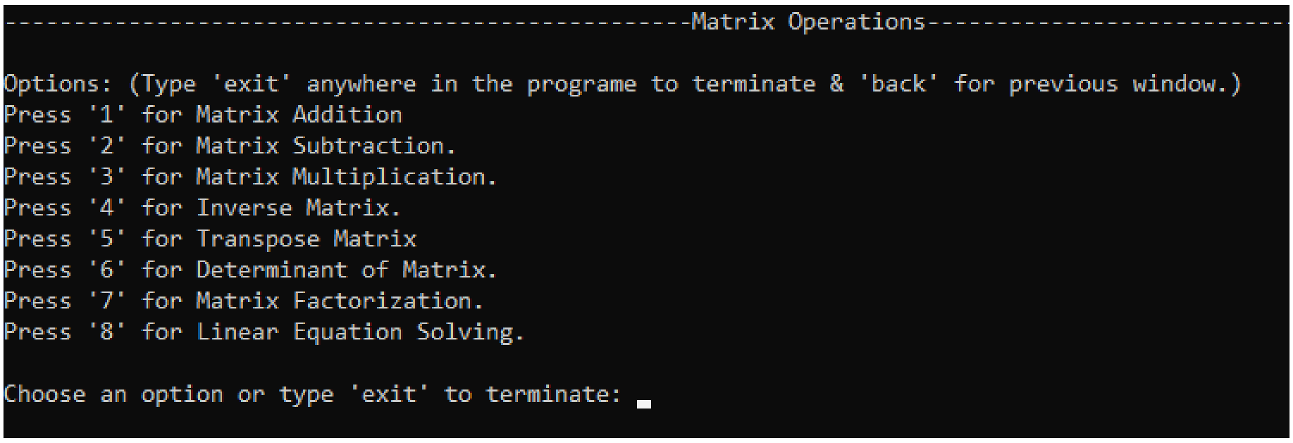
4.9) **Determinant:** Here I calculated the determinant. I used recursion to find it. At first, the inputted matrix is sent to a function which reduce the matrix by the row and column of the present cell and again find the determinant until a 2x2 matrix is got.

4.10) **Transpose Matrix:** Here I take an input matrix and pass the value of column to a value of row. For this, I took a matrix which stores the values of column into the values of rows of inputted matrix and vice-versa.

4.11) **Linear Equation Solving:** This is another important feature of my matrix operations. Here I solve n different equations using matrix. I used **Gauss-Seidel** to solve it. In **Gauss-Seidel,** I at first find if the matrix made by the coefficients of the left side of the equations are **Diagonally Dominant** or not. Then, I initialize the values of the variables with 0 and then I update those values at every steps. For example: Think I have 3 variables x, y, z. At first, x=0, y=0 & z=0. So, in the first iteration, I find the actual value of x using the first equation. In the equation I use the value of x and y. Now, I have value of x. In the second Equation I use the values of x & z and find y. So, now I have values of x & y. Then, in the third equation, I use values of x & y to find z. After these 3 steps are done, I find the error as I had stored the old values of x, y, z. So, I find the **Euclidean Distance** between old & new x, y, z & take the max difference which is known as my error. After several steps, I find the error get less than a particular epsilon value and thus I find the answer.

5. User Interface

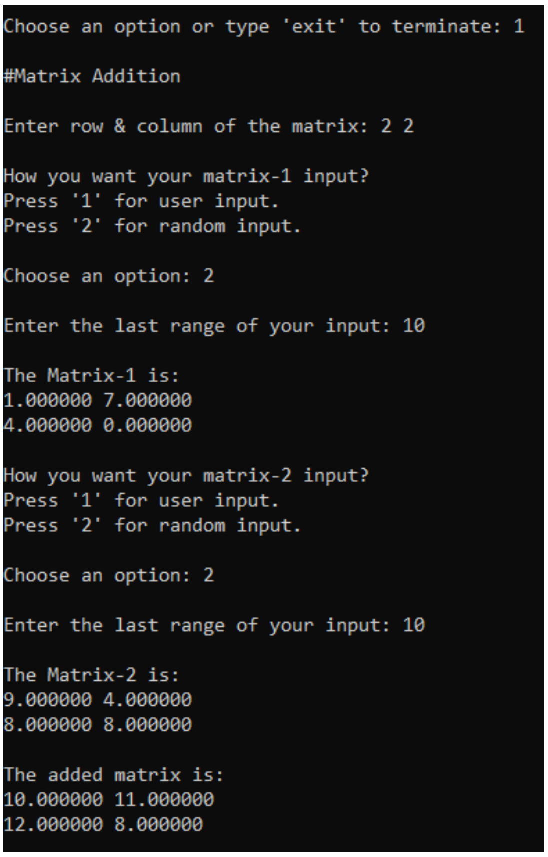
After Opening the console window, we get to choose options that which operation of matrix we want to do.



```
-------------------------------------------------Matrix Operations---------------------------

Options: (Type 'exit' anywhere in the programe to terminate & 'back' for previous window.)
Press '1' for Matrix Addition
Press '2' for Matrix Subtraction.
Press '3' for Matrix Multiplication.
Press '4' for Inverse Matrix.
Press '5' for Transpose Matrix
Press '6' for Determinant of Matrix.
Press '7' for Matrix Factorization.
Press '8' for Linear Equation Solving.

Choose an option or type 'exit' to terminate:
```

Figure 5.1: Main function

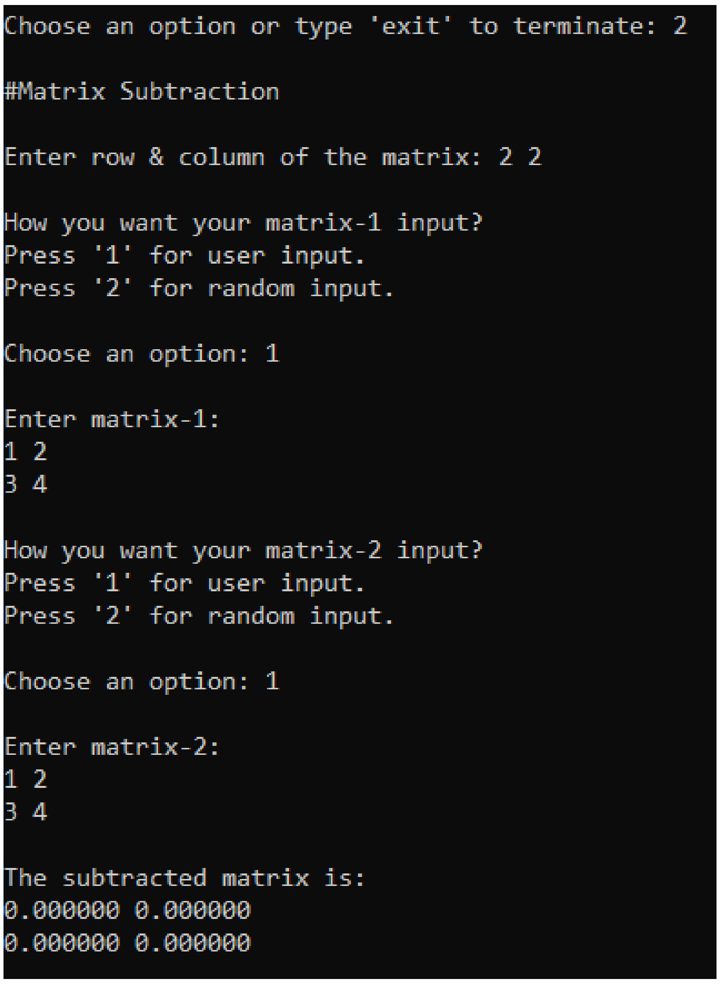
Now, if we select option '1', it will take us to matrix addition part & will take the dimension of the matrix as input. Then, we get two options. '1' for user input & '2' for random input. If we go to user input it will take the matrices and input and will show us the results. Else it will take us to the range input, which means the range of the elements which will be inputted. And if we take random inputs, it will show the inputted matrix as output after every input process is done. And at last, it adds two matrices and show us the result as output.



```
Choose an option or type 'exit' to terminate: 1

#Matrix Addition

Enter row & column of the matrix: 2 2

How you want your matrix-1 input?
Press '1' for user input.
Press '2' for random input.

Choose an option: 2

Enter the last range of your input: 10

The Matrix-1 is:
1.000000 7.000000
4.000000 0.000000

How you want your matrix-2 input?
Press '1' for user input.
Press '2' for random input.

Choose an option: 2

Enter the last range of your input: 10

The Matrix-2 is:
9.000000 4.000000
8.000000 8.000000

The added matrix is:
10.000000 11.000000
12.000000 8.000000
```

Figure 5.2: Matrix Addition

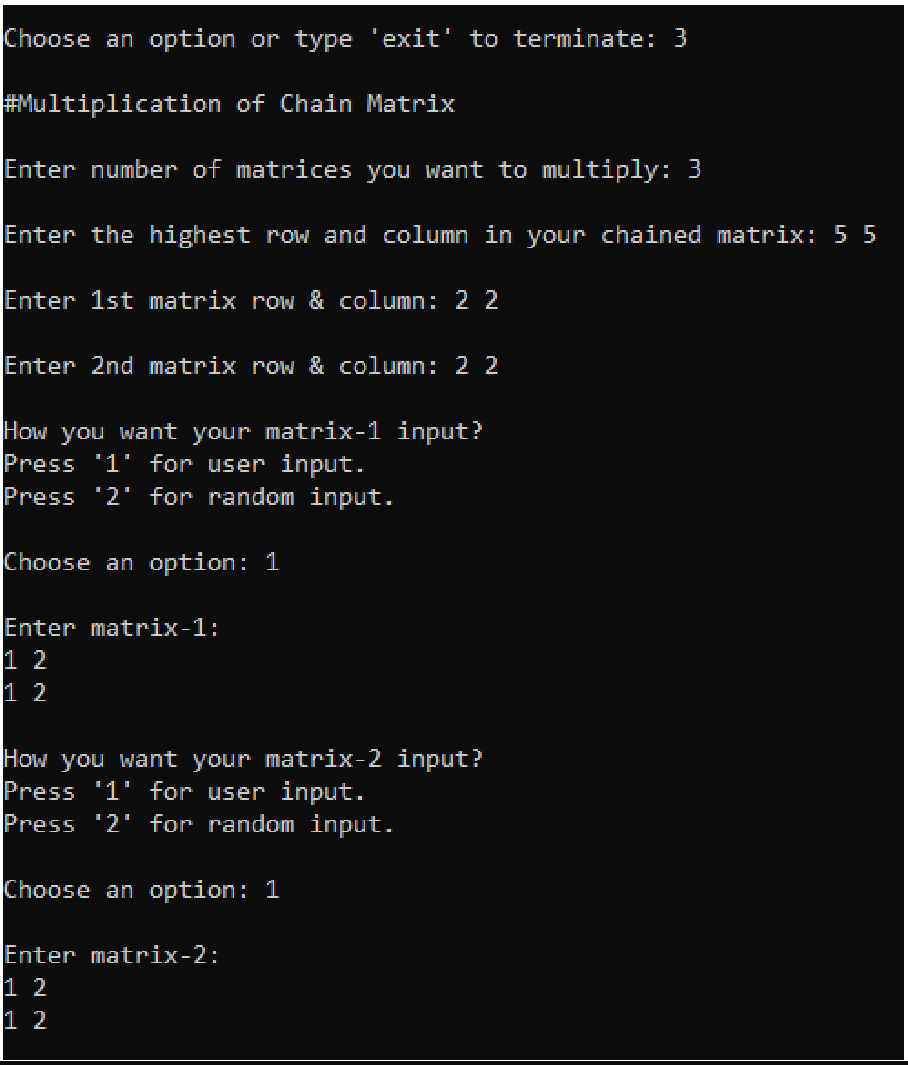
Now, if we select option ‘2’, it will take us to matrix subtraction part & will take the dimension of the matrix as input. Then, we get two options. ‘1’ for user input & ‘2’ for random input. If we go to user input it will take the matrices and input and will show us the results. Else it will take us to the range input, which means the range of the elements which will be inputted. And if we take random inputs, it will show the inputted matrix as output after every input process is done. And at last, it subtracts two matrices and show us the result as output.



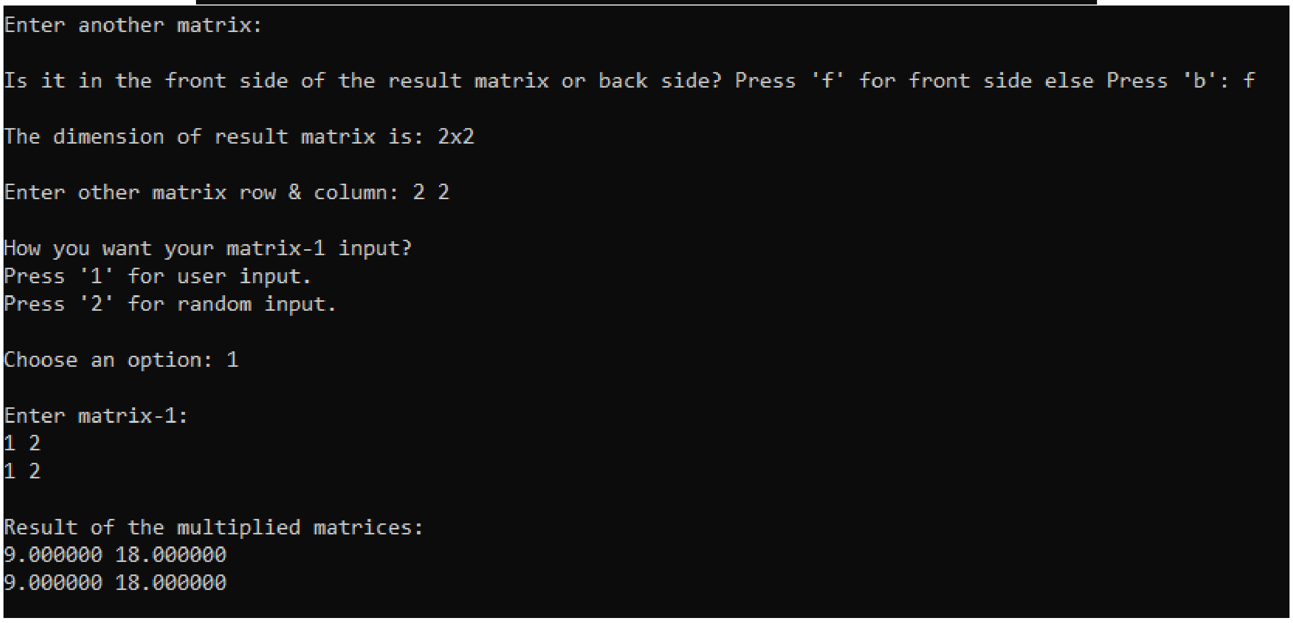
```
Choose an option or type 'exit' to terminate: 2

#Matrix Subtraction

Enter row & column of the matrix: 2 2

How you want your matrix-1 input?
Press '1' for user input.
Press '2' for random input.

Choose an option: 1

Enter matrix-1:
1 2
3 4

How you want your matrix-2 input?
Press '1' for user input.
Press '2' for random input.

Choose an option: 1

Enter matrix-2:
1 2
3 4

The subtracted matrix is:
0.000000 0.000000
0.000000 0.000000
```

Figure 5.3: Matrix Subtraction

Now, if we select option ‘3’, it will take us to matrix multiplication part & will take the dimension of the matrix as input. Here, before every matrix input, we get two options. ‘1’ for user input & ‘2’ for random input. If we go to user input it will take the matrices and input and will show us the results. Else it will take us to the range input, which means the range of the elements which will be inputted. And if we take random inputs, it will show the inputted matrix as output after every input process is done. Now, in here, at first, we take how much matrix a user wants to multiply. Then, we take the highest row and highest column he has in the chain matrix. Then we take dimensions of the matrix. After that, we take the matrix input. At the first time we take two matrix input and after that, every time we take one matrix as input before the chain ends. Before taking the Matrix input, we give and user two choice. When a matrix is to be multiplied with a result matrix it can be in two sides; at front of the result matrix and at backside of the result matrix. If user press ’f’, it will take the input matrix as matrix-1 and the previous multiplied matrix as matrix-2, and if pressed ‘b’ then the vice-versa happens.



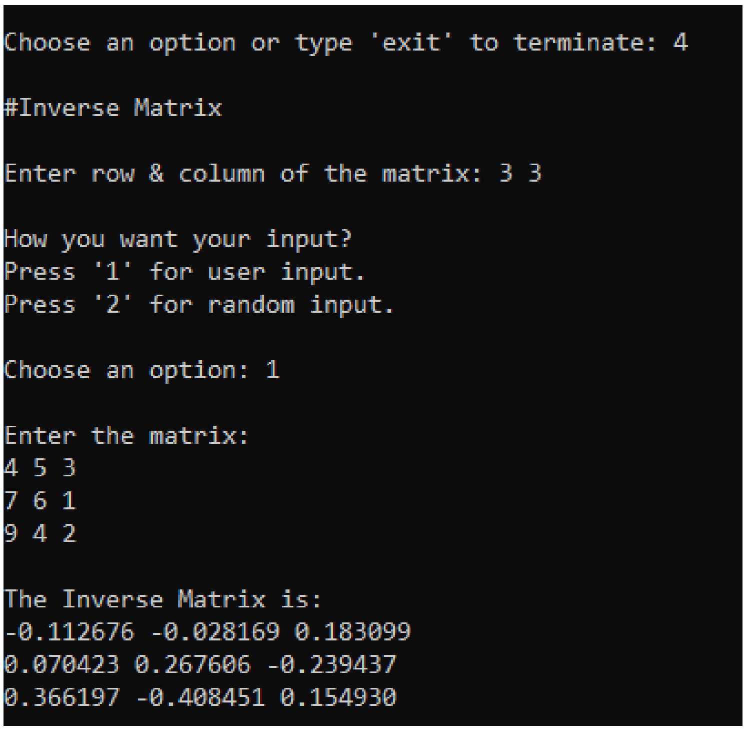
```
Choose an option or type 'exit' to terminate: 3

#Multiplication of Chain Matrix

Enter number of matrices you want to multiply: 3

Enter the highest row and column in your chained matrix: 5 5

Enter 1st matrix row & column: 2 2

Enter 2nd matrix row & column: 2 2

How you want your matrix-1 input?
Press '1' for user input.
Press '2' for random input.

Choose an option: 1

Enter matrix-1:
1 2
1 2

How you want your matrix-2 input?
Press '1' for user input.
Press '2' for random input.

Choose an option: 1

Enter matrix-2:
1 2
1 2
```



```
Enter another matrix:

Is it in the front side of the result matrix or back side? Press 'f' for front side else Press 'b': f

The dimension of result matrix is: 2x2

Enter other matrix row & column: 2 2

How you want your matrix-1 input?
Press '1' for user input.
Press '2' for random input.

Choose an option: 1

Enter matrix-1:
1 2
1 2

Result of the multiplied matrices:
9.000000 18.000000
9.000000 18.000000
```

Figure 5.4: Matrix Multiplication

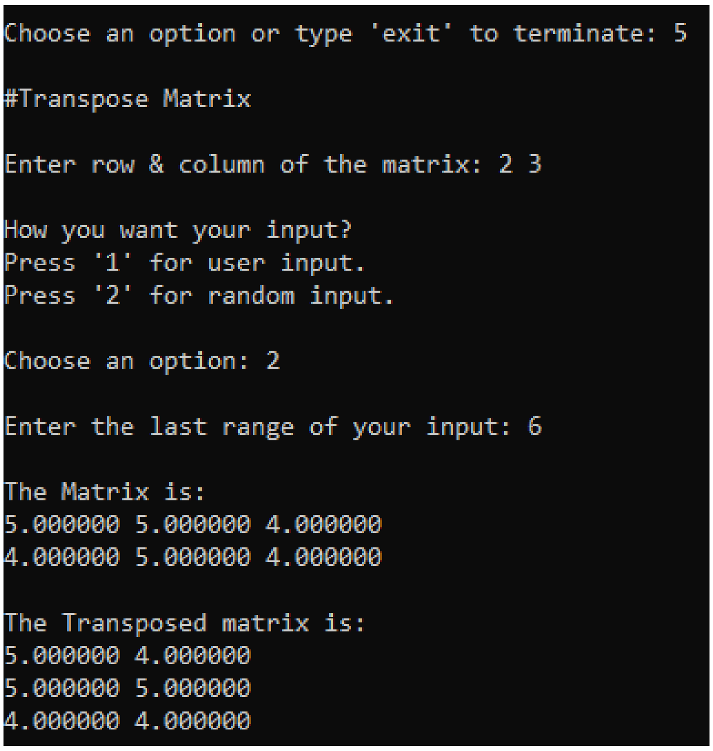
Now, if we select option ‘4’, it will take us to matrix inversion part & will take the dimension of the matrix as input. Here, before every matrix input, we get two options. ‘1’ for user input & ‘2’ for random input. If we go to user input it will take the matrices and input and will show us the results. Else it will take us to the range input, which means the range of the elements which will be inputted. And if we take random inputs, it will show the inputted matrix as output after every input process is done. Now, here at first it will take the dimension of the input matrix. Then, it will take the matrix as input. And then, it will calculate the inverse matrix and will show us the inverse matrix as output.



```
Choose an option or type 'exit' to terminate: 4

#Inverse Matrix

Enter row & column of the matrix: 3 3

How you want your input?
Press '1' for user input.
Press '2' for random input.

Choose an option: 1

Enter the matrix:
4 5 3
7 6 1
9 4 2

The Inverse Matrix is:
-0.112676 -0.028169 0.183099
0.070423 0.267606 -0.239437
0.366197 -0.408451 0.154930
```

Figure 5.5: Inverse Matrix

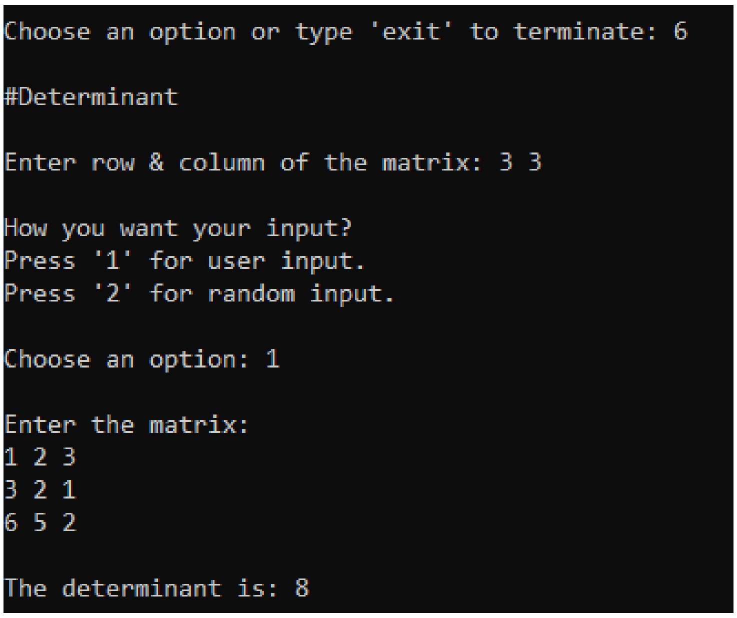
Now, if we select option ‘5’, it will take us to transpose matrix part & will take the dimension of the matrix as input. Here, before every matrix input, we get two options. ‘1’ for user input & ‘2’ for random input. If we go to user input it will take the matrices and input and will show us the results. Else it will take us to the range input, which means the range of the elements which will be inputted. And if we take random inputs, it will show the inputted matrix as output after every input process is done. Now, here at first it will take the dimension of the input matrix. Then, it will take the matrix as input. And then, it will transpose the matrix and will show us the transposed matrix as output.



```
Choose an option or type 'exit' to terminate: 5

#Transpose Matrix

Enter row & column of the matrix: 2 3

How you want your input?
Press '1' for user input.
Press '2' for random input.

Choose an option: 2

Enter the last range of your input: 6

The Matrix is:
5.000000 5.000000 4.000000
4.000000 5.000000 4.000000

The Transposed matrix is:
5.000000 4.000000
5.000000 5.000000
4.000000 4.000000
```

Figure 5.6: Transpose Matrix

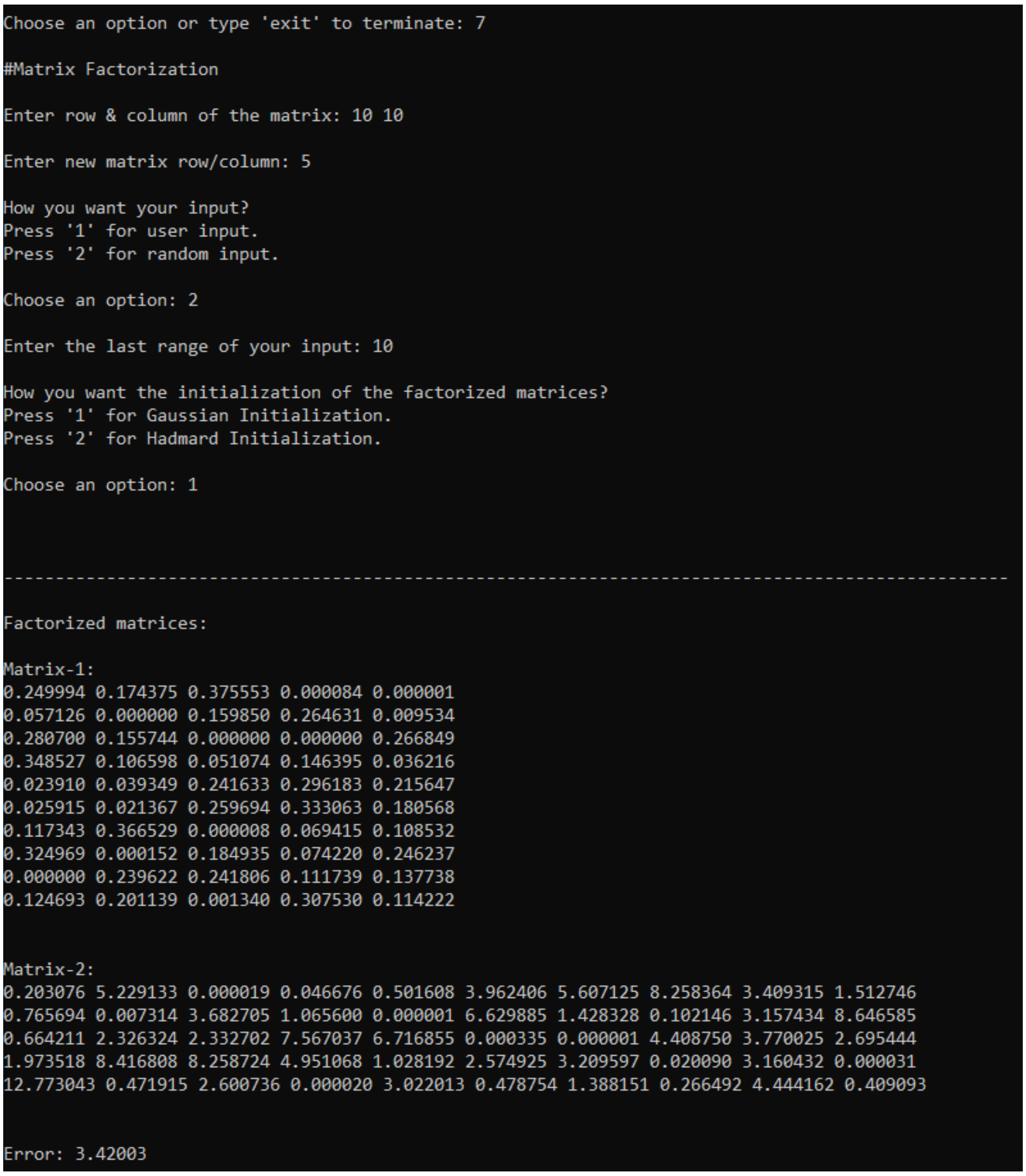
Now, if we select option ‘6’, it will take us to finding the determinant of a matrix part & will take the dimension of the matrix as input. Here, before every matrix input, we get two options. ‘1’ for user input & ‘2’ for random input. If we go to user input it will take the matrices and input and will show us the results. Else it will take us to the range input, which means the range of the elements which will be inputted. And if we take random inputs, it will show the inputted matrix as output after every input process is done. Now, here at first it will take the dimension of the input matrix. Then, it will take the matrix as input. And then, it will determinant of the matrix and will show us the determinant as output.



```
Choose an option or type 'exit' to terminate: 6

#Determinant

Enter row & column of the matrix: 3 3

How you want your input?
Press '1' for user input.
Press '2' for random input.

Choose an option: 1

Enter the matrix:
1 2 3
3 2 1
6 5 2

The determinant is: 8
```

Figure 5.7: Determinant Finding

Now, if we select option ‘7’, it will take us to matrix factorization part & will take the dimension of the matrix as input. Here, before every matrix input, we get two options. ‘1’ for user input & ‘2’ for random input. If we go to user input it will take the matrices and input and will show us the results. Else it will take us to the range input, which means the range of the elements which will be inputted. And if we take random inputs, it will show the inputted matrix as output after every input process is done. Now, here at first it will take the dimension of the input matrix. Then, it will take the reduced dimension as input and then, it will take the matrix as input. And then, it will determinant of the matrix and will show us the determinant as output. After that, it will ask us, how we want to initialize our reduced dimension matrices. If we press ‘1’, it will initialize them with gaussian initializer & if we press ‘2’, it will initialize them will Hadamard matrix initialization. Then after factorization, it will print the factorized matrices.



```
Choose an option or type 'exit' to terminate: 7

#Matrix Factorization

Enter row & column of the matrix: 10 10

Enter new matrix row/column: 5

How you want your input?
Press '1' for user input.
Press '2' for random input.

Choose an option: 2

Enter the last range of your input: 10

How you want the initialization of the factorized matrices?
Press '1' for Gaussian Initialization.
Press '2' for Hadmard Initialization.

Choose an option: 1



--------------------------------------------------------------------------------------------------

Factorized matrices:

Matrix-1:
0.249994 0.174375 0.375553 0.000084 0.000001
0.057126 0.000000 0.159850 0.264631 0.009534
0.280700 0.155744 0.000000 0.000000 0.266849
0.348527 0.106598 0.051074 0.146395 0.036216
0.023910 0.039349 0.241633 0.296183 0.215647
0.025915 0.021367 0.259694 0.333063 0.180568
0.117343 0.366529 0.000008 0.069415 0.108532
0.324969 0.000152 0.184935 0.074220 0.246237
0.000000 0.239622 0.241806 0.111739 0.137738
0.124693 0.201139 0.001340 0.307530 0.114222


Matrix-2:
0.203076 5.229133 0.000019 0.046676 0.501608 3.962406 5.607125 8.258364 3.409315 1.512746
0.765694 0.007314 3.682705 1.065600 0.000001 6.629885 1.428328 0.102146 3.157434 8.646585
0.664211 2.326324 2.332702 7.567037 6.716855 0.000335 0.000001 4.408750 3.770025 2.695444
1.973518 8.416808 8.258724 4.951068 1.028192 2.574925 3.209597 0.020090 3.160432 0.000031
12.773043 0.471915 2.600736 0.000020 3.022013 0.478754 1.388151 0.266492 4.444162 0.409093


Error: 3.42003
```

Figure 5.8: Matrix Factorization

Now, if we select option ‘8’, it will take us to linear equation solving part. Here, before every matrix input, we get two options. ‘1’ for user input & ‘2’ for random input. If we go to user input it will take the matrices and input and will show us the results. Else it will take us to the range input, which means the range of the elements which will be inputted. And if we take random inputs, it will show the inputted matrix as output after every input process is done. Now, here at first it will take the number of variables. Then, it will take the left-side coefficient matrix as input. After that, it will take the right-side values input. And then, it will calculate using gauss-seidel and will print the answers of the variables. And one thine thing is noticeable here. If the coefficient matrix is not diagonally dominant then it will say the there is no solution.

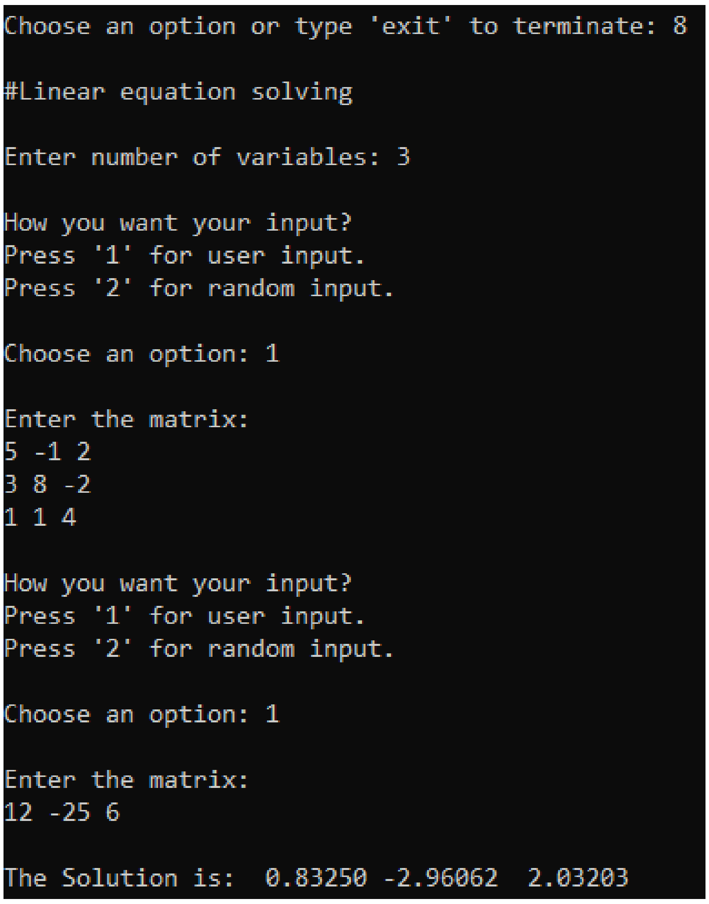

```
Choose an option or type 'exit' to terminate: 8

#Linear equation solving

Enter number of variables: 3

How you want your input?
Press '1' for user input.
Press '2' for random input.

Choose an option: 1

Enter the matrix:
5 -1 2
3 8 -2
1 1 4

How you want your input?
Press '1' for user input.
Press '2' for random input.

Choose an option: 1

Enter the matrix:
12 -25 6

The Solution is:  0.83250 -2.96062  2.03203
```

Figure 5.9: Linear equation solving

6. Challenges Faced

- Finding the actual algorithms of multiplicative update.
- Understanding the derivations of the algorithms.
- Working with big data and handling it.
- Making a user-friendly program.
- Factorization was not working for some inputted matrix, handling was a big challenge.
- Working with header file for the first time.
- Working with more than 1 C++ files.
- Passing a 2D matrix in a function and getting the result from it.
- Allocating memory for every matrix was quite a challenge.
- Handling large code for the first time.
- Implementing mathematical notations in code.

7. Conclusion

From the project, I have learnt the operations of matrix. I have the algorithms, which can help me in future studies. I have learnt handling big data and big code & multiple CPP files. I have also learnt how to make a user-friendly program.

My future plan is to extend this project. I want to make a graphical interface for this. Beside this, I want to add all the algorithms of matrix factorization here and check which one is more efficient. Most importantly, I want to make an application which is used for matrix factorization.

Reference

https://mathworld.wolfram.com/ Wolfram MathWorld 29-05-2022

https://www.youtube.com/ Youtube 29-05-2022

https://www.wikipedia.org/ Wilipedia 29-05-2022

Burred, Juan José. "Detailed derivation of multiplicative update rules for NMF." *Paris, France* (2014).

Lipschutz, Seymour, and Marc Lars Lipson. *Schaum's Outline of Linear Algebra*. McGraw-Hill Education, 2013.

Petersen, Kaare Brandt, and Michael Syskind Pedersen. "The matrix cookbook." *Technical University of Denmark* 7.15 (2008): 510.