UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

ROBOTICS ENGENEERING DEPARTMENT

BASIC SCIENCE CENTER

Representing Linear Algebra in C

Mathematics Keyboard '75

Miguel Angel Romero Hernández

Marco Arturo Nuñez Michaus

Programming Logic

Eduardo Emmanuel Rodríguez López, EngD.

June 22, 2020

UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

ROBOTICS ENGENEERING DEPARTMENT

Basic Science Center. June 22, 2020.

Mathematics Keyboard '75

B.E. M.A. Michaus, B.E. M.A. Romero

Email: al271989@edu.uaa.mx, al278677@edu.uaa.mx

Abstract: Linear algebra is fundamental to robot modeling, compute axial movements, avoid obstacles, correct joint angles and more. This document is aimed to provide a set of solutions to compute all major processes, methods and rules from linear algebra using GCC's C language and compiler.

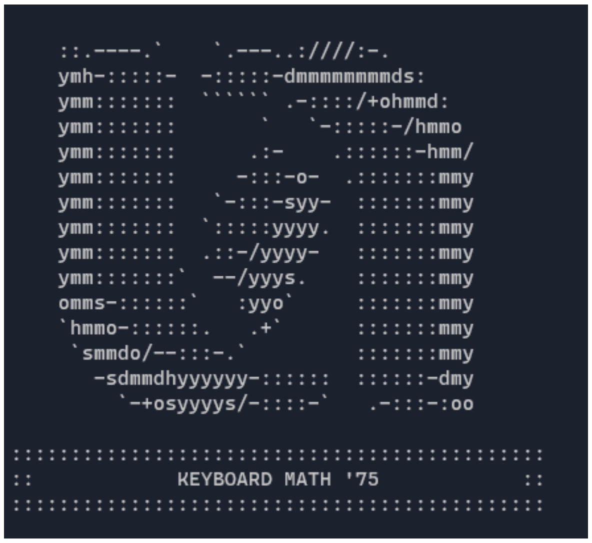
Linear Algebra

Pronoun, f.

The branch of mathematics concerning linear equations such as. linear maps such as. and their representations in vector spaces and through matrices. Linear algebra is central to almost all areas of mathematics.

Keywords:

Linear, algebra, robot, Gauss, Crammer, Sarrus, elimination, backward, substitution, transposition, dot, summation, product, matrix.



```
    ::.----.`    `.---..:////:-.
    ymh-:::::-  -:::::-dmmmmmmmmds:
    ymm:::::::  `````` .-::::/+ohmmd:
    ymm:::::::       `   `-:::::-/hmmo
    ymm:::::::      .:-     .::::::-hmm/
    ymm:::::::     -:::-o-  .:::::::mmy
    ymm:::::::   `-:::-syy-  :::::::mmy
    ymm:::::::  `:::::yyyy.  :::::::mmy
    ymm:::::::  .::-/yyyy-   :::::::mmy
    ymm:::::::`  --/yyys.    :::::::mmy
    omms-::::::`   :yyo`     :::::::mmy
    `hmmo-::::::.    .+`     :::::::mmy
     `smmdo/--:::-.`         :::::::mmy
       -sdmmdhyyyyyy-::::::  ::::::-dmy
         `-+osyyyys/-::::-`   .-:::-:oo

:::::::::::::::::::::::::::::::::::::::::::::
::            KEYBOARD MATH '75            ::
:::::::::::::::::::::::::::::::::::::::::::::
```

Figure 1: The official logo of Keyboard Math '75

1.1 Introduction

This document is an extension to the program "Mathematics Keyboard '75" (also referred to as MK75) a product written in GCC's C programming language. Even tough, an executable is provided to prove able to compute all mayor linear-algebraic algorithms, and operate under any hardware supporting a compilated program written in C; this program for the most part is aimed to be tested singularly for each of the algorithms provided within as an alternate solution to current algorithms as present solutions have limited readability, and their times of execution are mostly not properly optimized.[1]

Initially the reader will be provided with a set of brief definitions that are necessary to allow the understanding of this paper (literature review) and therefore able to operate and modify any of the files adjacent to this document without causing errors. Next, a set of instructions will be rendered to the reader to compile and later open an instance of the program; Meanwhile, this document will serve to assist the newly found user of said program, Mathematics Keyboard '75. Finally, a conclusion will be drawn to display the importance of both: this document and the program to all: the national industry, those in pursue of a better functioning robot, and readers interested in computer science. And as a reference high quality copies of all screen shoots will be provided at the end of this document.

1.2 GCC'S C

GCC's programing language will referred throughout this document simply as C. C was invented as a general-purpose procedural solution supporting structured syntax and a lexical variable scope and recursion, and that by definition c is a mid-level language which refers to languages that bind the gap between a machine-level language and high-level language. Thus, having better performance at execution time than high-level or interpreted languages, and better code readability than machine level languages. [2]

It is of up most importance for the reader to have previous knowledge (before the reading of this work) and a basic understanding of GCC's C programming language and the good practices associated to C programs.

1.3 Linear Algebra

The branch of mathematics that studies the concepts behind linear equations, systems of linear equations, bidimensional space, vectors, arrays, matrixes. Linear algebraic algorithms are continuously being used by national engineers to compute kinematic equations and calculate axial or angular movements at runtime. [3]

This document provides insights and the logical representation in C for all mayor linear algebraic algorithms. The executable MK75 is optimized for testing the computations of data from square matrixes of n x n size (by default the maximum size is fixed to 10).

- Gaussian elimination
- Crammer's rule
- Sarrus' method
- Summation of two Matrixes
- Dot Product of two Matrixes

Although the logic behind all algorithms represented in code within the MK75 program will be briefly explained during this literature review it is necessary that the reader understands the logic and rules behind those mathematical algorithms, as well as their deficiencies and limitations.

1.4 Gaussian Elimination

A process also known as row reduction in linear algebra, is an algorithm for solving systems of equations. This process, invented by Carl Friedrich Gauss, is a set of operations performed as a sequence. This algorithm's operations have the objective of transforming any matrix into its upper triangular form (row echelon form). [4]

The operations involved within this algorithm are row swapping, multiplication of a row by a non-zero number, adding a multiple of one row to another. The following system of equations (three equations containing three variables) can be solved using Gaussian Elimination and then performing backward substitution). [4]

$$\begin{alignedat}{4} 2x &+ y &- z &= 8 \\ -3x &- y &+ 2z &= -11 \\ -2x &+ y &+ 2z &= -3 \end{alignedat}$$

Formula 1: A system of equations containing three equations and three variables from Wikimedia Foundation.

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ -3 & -1 & 2 & -11 \\ -2 & 1 & 2 & -3 \end{array}\right]$$

Formula 2: A system of equations from Fig 1 in matrix for from Wikimedia Foundation.

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 8 \\ 0 & \frac{1}{2} & \frac{1}{2} & 1 \\ 0 & 0 & -1 & 1 \end{array}\right]$$

Formula 3: The matrix from Fig. 2 transformed into its triangular form (echelon from) from Wikimedia Foundation.

Although, gaussian eliminations is used as an algorithm to solve systems, there are a few things that will prevent a computer from being able to solve a system of equation, like inconsistencies inside a system of equations or intercepting equations causing infinite solutions. Nevertheless, this can be detected by using pivotal properties of matrixes.[4]

1.5 Cramer's Rule

Cramer's rule is referred to a theorem that finds the value of a set of variables involved in a system of equations with as many variables as there are equations. The creator of this theorem is Gabriel Cramer and this rule is presented as a way of using determinants to solve, as explained before, a system of equations. [5]

This theorem is inefficient by design for industrial equipment with low computing power, as the algorithm will calculate a determinant of a matrix and a determinant for each variable (done by swapping the vector of constants and the variable's column).

$$x_i = \frac{\det(A_i)}{\det(A)} \qquad i = 1, \ldots, n$$

Formula. 4: Cramer's rule calculates the value of each variable from matrix A. The value of x_i is the product of det(A_i)(detA)$^{-1}$ from Wikimedia Foundation.

1.6 Rule of Sarrus

Sarrus' rule is used to calculate the determinant of a squared matrix. This method is a geometrical approach for finding a determinant of a matrix of specific sizes and ranks (the geometric relation and signs can vary).[6]

$$\det(A) = \sum_{\tau \in S_n} \operatorname{sgn}(\tau) \prod_{i=1}^{n} a_{i,\tau(i)} = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} a_{\sigma(i),i},$$

Formula 5: Leibniz formula, an algebraic approach to the rule of Sarrus from Wikimedia Foundation.

Within this document, the rule of Sarrus is specific to matrixes of size 3 or 2, due to the increment of complexity of this algorithm and the prone capacity of this theorem to fail while calculating for matrixes of different sizes and ranks.

1.7 Linear Algebra and the definition of Matrix Multiplication and a Dot Product Operation

Although, in linear algebra a dot product operation is known as the scalar product of two vectors. When operating with matrixes or lattices this operation is then known as a matrix multiplication. In linear algebra during a **matrix multiplication**, each entry in the product matrix is the **dot product** of a row in the first matrix and a column in the second matrix.[7]

$$\mathbf{A} \cdot \mathbf{B} = \begin{bmatrix} A_1 & A_2 & \ldots & A_n \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \cdot \\ \cdot \\ \cdot \\ B_n \end{bmatrix} = A_1B_1 + A_2B_2 + \ldots + A_nB_n$$

Formula 6: An algebraic approach to the calculation of two vectors dot product from Wikimedia Foundation.

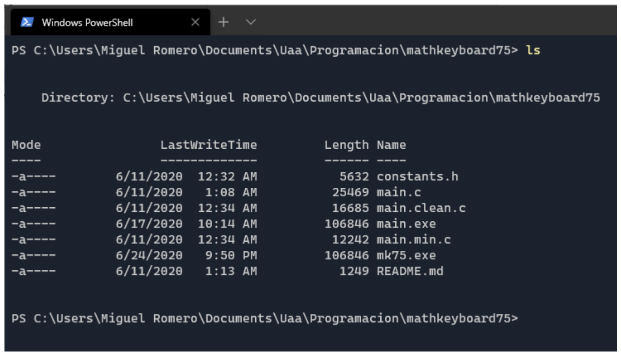
$$\begin{array}{cccc} & & \begin{array}{cc}\vec{b_1} & \vec{b_2}\\ \downarrow & \downarrow\end{array} & \\ \begin{array}{c}\vec{a_1}\rightarrow\\ \vec{a_2}\rightarrow\end{array} & \begin{bmatrix}1&7\\2&4\end{bmatrix} \cdot & \begin{bmatrix}3&3\\5&2\end{bmatrix} = & \begin{bmatrix}\vec{a_1}\cdot\vec{b_1} & \vec{a_1}\cdot\vec{b_2}\\ \vec{a_2}\cdot\vec{b_1} & \vec{a_2}\cdot\vec{b_2}\end{bmatrix}\\ & A & B & C\end{array}$$

Formula 7: An algebraic approach to a matrix multiplication resulting in a Matrix C from Khan Academy.

Moreover, as recognized by mathematicians, even though that a matrix multiplication is a series of dot product operations, these are two different operations with different purposes.

2.1 Compiling and Start

First, provided that the user is at the working directory and that all program files are present (see Figure 2), the user can open the command terminal, again, pointing to this directory.



```
PS C:\Users\Miguel Romero\Documents\Uaa\Programacion\mathkeyboard75> ls


    Directory: C:\Users\Miguel Romero\Documents\Uaa\Programacion\mathkeyboard75


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         6/11/2020  12:32 AM           5632 constants.h
-a----         6/11/2020   1:08 AM          25469 main.c
-a----         6/11/2020  12:34 AM          16685 main.clean.c
-a----         6/17/2020  10:14 AM         106846 main.exe
-a----         6/11/2020  12:34 AM          12242 main.min.c
-a----         6/24/2020   9:50 PM         106846 mk75.exe
-a----         6/11/2020   1:13 AM           1249 README.md


PS C:\Users\Miguel Romero\Documents\Uaa\Programacion\mathkeyboard75>
```

Figure 2: User's terminal pointed at the working directory; All program files are present.

Second, depending on what system the user is using a second terminal might or might not open upon running the last command. [8]

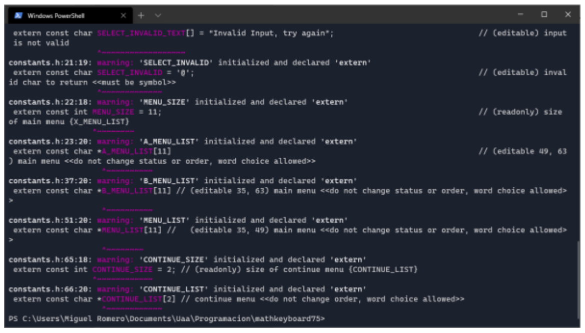
To run the compiled program simply run:

```
Windows$ ./mk75.exe
MacOS$   ~/mk75.out
//Linux distros should be exempt
```

Finally, the user can close the first terminal (previously used to compile the program), although this is not necessary and not always the case for systems that fail to open a second terminal. From here on now we will be referring only to the terminal running the program MK75 (the terminal displaying the title of the program). If all steps were correctly performed the program is now compiled and running.

Run the following command to compile and execute the MK75 program:

```
gcc main.min.c -o mk75
```

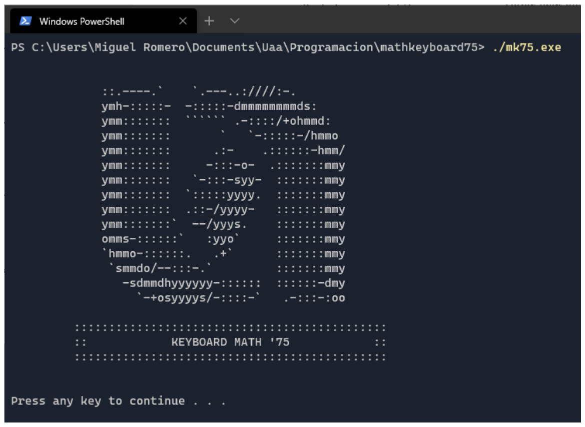


```
 extern const char SELECT_INVALID_TEXT[] = "Invalid Input, try again";                    // (editable) input
 is not valid
                   ^~~~~~~~~~~~~~~~~~~
constants.h:21:19: warning: 'SELECT_INVALID' initialized and declared 'extern'
 extern const char SELECT_INVALID = '@';                                                  // (editable) inval
id char to return <<must be symbol>>
                   ^~~~~~~~~~~~~~
constants.h:22:18: warning: 'MENU_SIZE' initialized and declared 'extern'
 extern const int MENU_SIZE = 11;                                                         // (readonly) size
of main menu {X_MENU_LIST}
                  ^~~~~~~~~
constants.h:23:20: warning: 'A_MENU_LIST' initialized and declared 'extern'
 extern const char *A_MENU_LIST[11]                                                       // (editable 49, 63
) main menu <<do not change status or order, word choice allowed>>
                    ^~~~~~~~~~~
constants.h:37:20: warning: 'B_MENU_LIST' initialized and declared 'extern'
 extern const char *B_MENU_LIST[11] // (editable 35, 63) main menu <<do not change status or order, word choice allowed>
>
                    ^~~~~~~~~~~
constants.h:51:20: warning: 'MENU_LIST' initialized and declared 'extern'
 extern const char *MENU_LIST[11] //   (editable 35, 49) main menu <<do not change status or order, word choice allowed>
>
                    ^~~~~~~~~
constants.h:65:18: warning: 'CONTINUE_SIZE' initialized and declared 'extern'
 extern const int CONTINUE_SIZE = 2; // (readonly) size of continue menu {CONTINUE_LIST}
                  ^~~~~~~~~~~~~
constants.h:66:20: warning: 'CONTINUE_LIST' initialized and declared 'extern'
 extern const char *CONTINUE_LIST[2] // continue menu <<do not change order, word choice allowed>>
                    ^~~~~~~~~~~~~
PS C:\Users\Miguel Romero\Documents\Uaa\Programacion\mathkeyboard75>
```

Figure 3: The expected output after the program is compiled, the working directory now holds the updated mk75 file.

3.1 Start of Execution

Throughout this work, the program's execution starts with a welcoming title just below a short piece of ACSII art. Depending on what operating system and its default language the program should instruct to **press any key to continue**. At this moment, the user should follow accordingly and press any key. (see Figure 4)

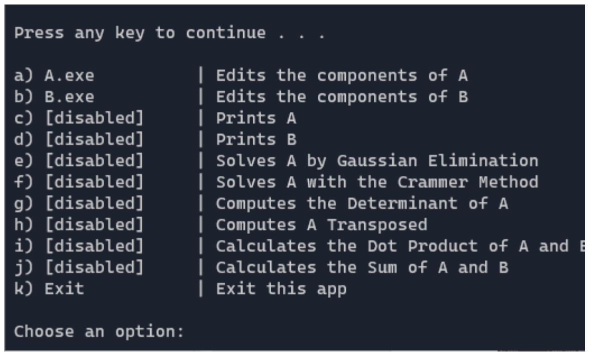


```
PS C:\Users\Miguel Romero\Documents\Uaa\Programacion\mathkeyboard75> ./mk75.exe


            ::.----.`    `.---..:////:-.
            ymh-:::::-  -:::::-dmmmmmmmmds:
            ymm:::::::  ``````  .-::::/+ohmmd:
            ymm:::::::        `   `-:::::-/hmmo
            ymm:::::::       .:-    .::::::-hmm/
            ymm:::::::      -:::-o-  .:::::::mmy
            ymm:::::::    `-:::-syy-  :::::::mmy
            ymm:::::::  `:::::yyyy.  :::::::mmy
            ymm:::::::  .::-/yyyy-   :::::::mmy
            ymm:::::::`  --/yyys.    :::::::mmy
            omms-::::::`    :yyo`     :::::::mmy
            `hmmo-::::::.    .+`      :::::::mmy
             `smmdo/--:::-.`          :::::::mmy
               -sdmmdhyyyyyy-::::::  ::::::-dmy
                 `-+osyyyys/-::::-`   .-:::-:oo

        ::::::::::::::::::::::::::::::::::::::::::::
        ::             KEYBOARD MATH '75          ::
        ::::::::::::::::::::::::::::::::::::::::::::


Press any key to continue . . .
```

Figure 4: The program is now running inside the system's terminal.

3.2 Main Menu

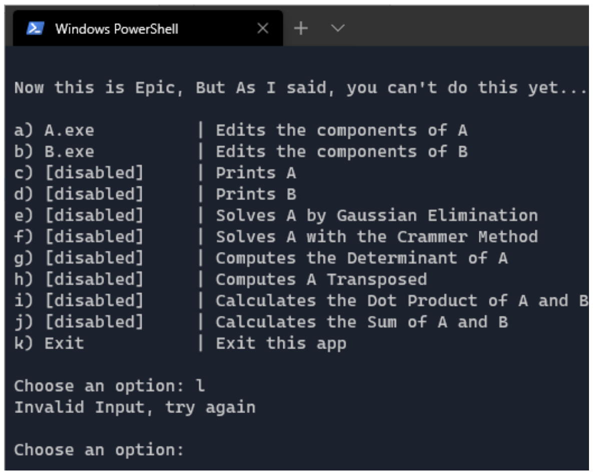
The program displays a different screen after a key is pressed ay the title screen, during the total length of this document the reference to this menu is made as the main menu. For now, the only enabled options are a), b), and k); however, a reference to the main menu will be arranged at any of its states, having three, eight or all options enabled.



```
Press any key to continue . . .

a) A.exe            | Edits the components of A
b) B.exe            | Edits the components of B
c) [disabled]       | Prints A
d) [disabled]       | Prints B
e) [disabled]       | Solves A by Gaussian Elimination
f) [disabled]       | Solves A with the Crammer Method
g) [disabled]       | Computes the Determinant of A
h) [disabled]       | Computes A Transposed
i) [disabled]       | Calculates the Dot Product of A and B
j) [disabled]       | Calculates the Sum of A and B
k) Exit             | Exit this app

Choose an option:
```

Figure 5: The Main Menu is displayed, some options are disabled, this is just after the user presses a key and therefore continues the execution.

Because, the program needs user data to do all other operations: if the user chooses a disabled option, the program will loop and only display "**Now this is epic, but as I said you can't do this yet…**"; however, if an inexistent option is chosen the program will react accordingly and log "**Invalid input, try again…**".



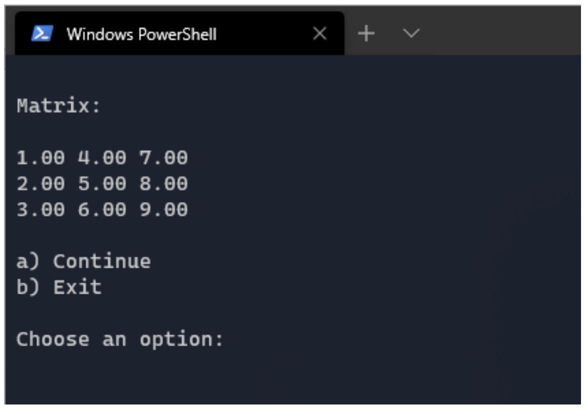
```
Now this is Epic, But As I said, you can't do this yet...

a) A.exe            | Edits the components of A
b) B.exe            | Edits the components of B
c) [disabled]       | Prints A
d) [disabled]       | Prints B
e) [disabled]       | Solves A by Gaussian Elimination
f) [disabled]       | Solves A with the Crammer Method
g) [disabled]       | Computes the Determinant of A
h) [disabled]       | Computes A Transposed
i) [disabled]       | Calculates the Dot Product of A and B
j) [disabled]       | Calculates the Sum of A and B
k) Exit             | Exit this app

Choose an option: l
Invalid Input, try again

Choose an option:
```

Figure 6: The system's terminal logs different outputs for inexistent and invalid options.

This validation holds true for all inexistent or disabled options throughout the program. If the option is valid the menu will then execute the logic related to that option.

3.2 Continue Menu

The program is designed to be tested over and over, so after each of the functions are completed the user will be prompted with the option to end the execution of MK75 or continue inside the loop and therefore going back to the main menu.

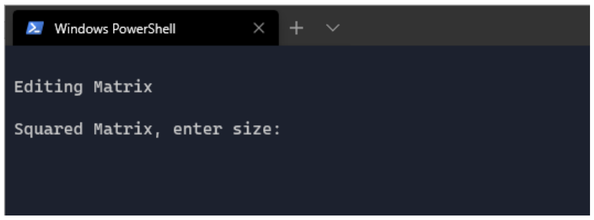


```
Matrix:

1.00 4.00 7.00
2.00 5.00 8.00
3.00 6.00 9.00

a) Continue
b) Exit

Choose an option:
```

Figure 7: The user is asked to continue after doing an operation as intended.

3.3 Matrix Editors

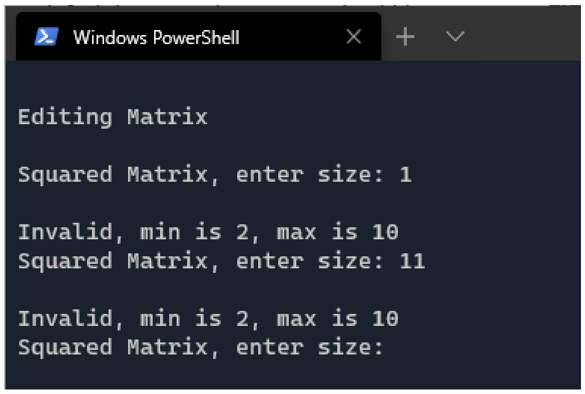
This document refers to editors to options a or b of the main menu, the MK75 by default has the capacity to run operations on two matrixes being matrix A and B, the most important matrix would be matrix A because it will be involved during any operation that only needs one matrix to be computed.



```
Windows PowerShell

Editing Matrix

Squared Matrix, enter size:
```

Figure 8: Upon selecting 'a) or b)' from the main menu, a text that reads 'Editing Matrix' is displayed, the user also is asked to enter the matrix size.

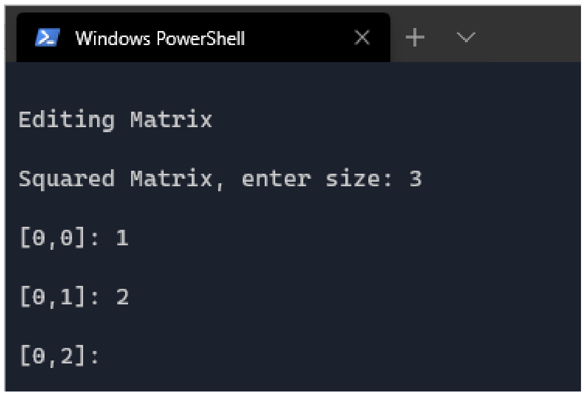
To edit any of the two matrixes A or B, the user can simply type "**a**" or "**b**" into the command terminal and send the instruction with the **Enter Key**. This is the same process to enter any data to the terminal, although some inputs validate characters others validate numbers (see Figure 8 and 10)



```
Windows PowerShell

Editing Matrix

Squared Matrix, enter size: 1

Invalid, min is 2, max is 10
Squared Matrix, enter size: 11

Invalid, min is 2, max is 10
Squared Matrix, enter size:
```

Figure 9: Expected output when the user inputs an invalid value.

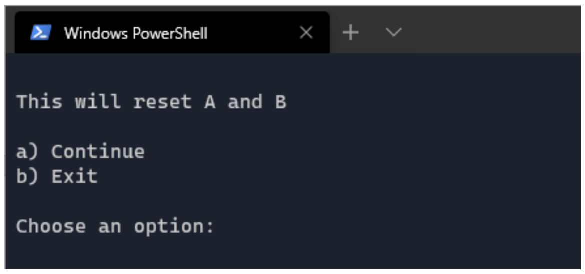
Now, the user is asked to enter a matrix size, this refers to the value that n holds when referring to an n * n matrix, also known as a squared matrix. By default, the program has a maximum size of 10 and a minimum of 2.



```
Windows PowerShell

Editing Matrix

Squared Matrix, enter size: 3

[0,0]: 1

[0,1]: 2

[0,2]:
```

Figure 10: When a valid matrix is read by the program the user is allowed to input matrix data for each index.

However, if the input is valid the program will allow the user to continue and now the user can write data to the MK75 program. (see Figure 10)



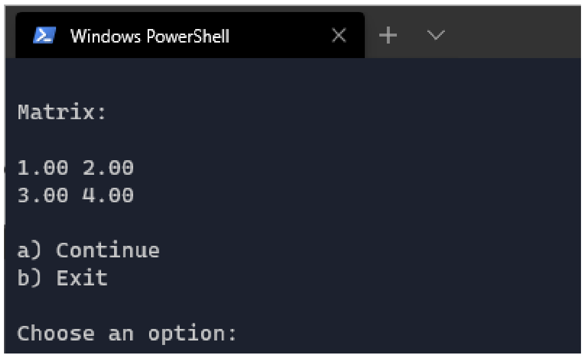
```
Windows PowerShell

This will reset A and B

a) Continue
b) Exit

Choose an option:
```

Figure 11: The user is warned before continuing and therefore changing matrix's data.

Furthermore, if the user enters the editor once more to edit a matrix that holds valid data, the user will have to reset both matrixes. And he will be able to input a new matrix size. If the user opts to not continue the action is skipped and the continue menu is displayed.

3.4 Matrix Printers

After the program detects to have valid data for matrix A or both: matrixes A and B, the user is presented with the option to check the values by printing them. The option to print matrix A or B is at the main menu labeled as options c) and d) respectively.



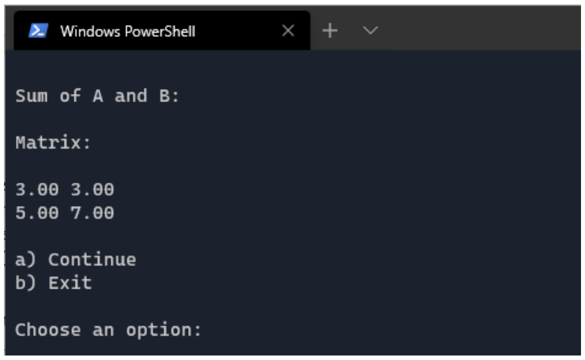
Windows PowerShell

```
Matrix:

1.00 2.00
3.00 4.00

a) Continue
b) Exit

Choose an option:
```

Figure 12: The command terminal logs data in matrix-form, this is just after the option to print matrix A or B is selected.

3.5 Matrix A and B Summation

Unlike, the other options the programs that perform operations will not alter any matrix data, instead the result will be calculated and then printed shortly after.



Windows PowerShell

```
Sum of A and B:

Matrix:

3.00 3.00
5.00 7.00

a) Continue
b) Exit

Choose an option:
```

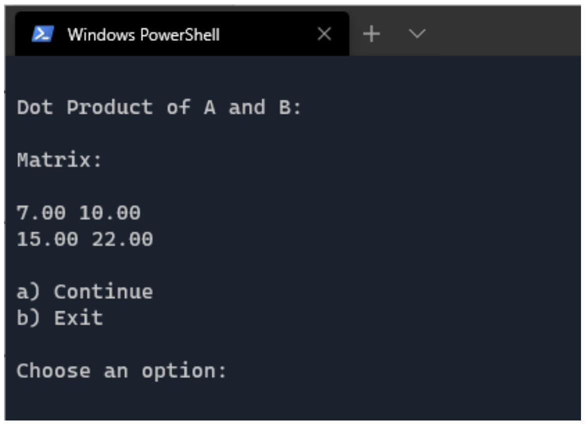
Figure 13: A sum of two matrixes is performed and then displayed in the command prompt.

Significantly during the whole extent of this paper, the reference to this process, from the options of the main menu, is stated to be connected to option j). Moreover, as titled above, this option performs an operation referred to in Linear Algebra as a sum of two matrixes.

Because this program's objective is the testing algorithms, at the conclusion of this operation, or any other operation that performs a form of calculation: the data of any of the matrixes A or B will have suffered no modifications at all, this is due to convenience when performing any other operations to matrix A or B.

3.6 Matrix A and B Multiplication (Dot Product)

Similarly, to the operation of matrix summation, this operation needs valid data from two matrixes to be performed, otherwise the option will be disabled as previously demonstrated. The goal of this part of the program, option i) of the main menu, is to calculate the matrix multiplication through a series of dot product operations of two matrixes.



Windows PowerShell

```
Dot Product of A and B:

Matrix:

7.00 10.00
15.00 22.00

a) Continue
b) Exit

Choose an option:
```

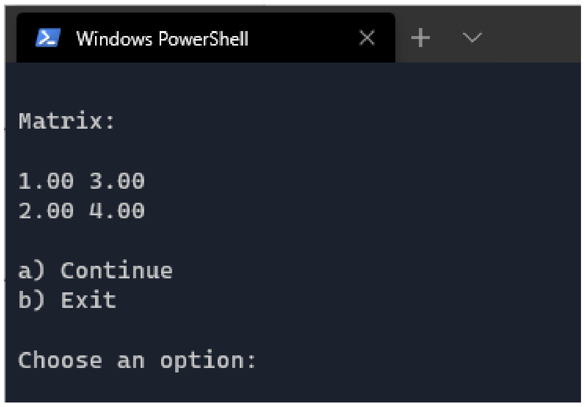
Figure 14: A multiplication of two matrixes is performed as the aftermath of a series of dot product operations.

3.7 Transposition of Matrix A

The programs 6th option, option h) of the main menu, corresponds to a matrix transpose operation, in linear algebra is referred to a matrix with mirrored indexes for each of its entries.[9]

$$[\mathbf{A}^{\mathrm{T}}]_{ij} = [\mathbf{A}]_{ji}.$$

Formula 6: An algebraic formal characteristic of transposed Matrixes, where a value with mirrored indexes is equal to the original Matrix's value.



```
Matrix:

1.00 3.00
2.00 4.00

a) Continue
b) Exit

Choose an option:
```

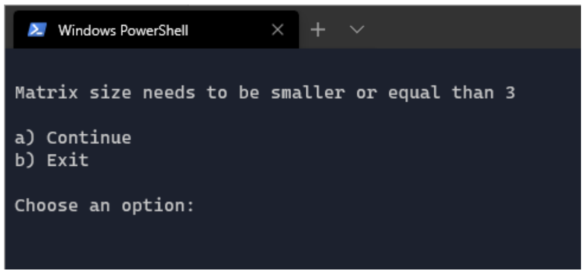
Figure 15: Matrix A is transposed; the resulting matrix is printed.

Undoubtedly, MK75 performs under the same principle, and in turn upon selecting to perform a transpose operation the program will calculate and execute said process using the data from matrix A and then shortly after logging the resulting matrix in the command prompt. (see Figure 15)

3.8 Determinants: Rule of Sarrus

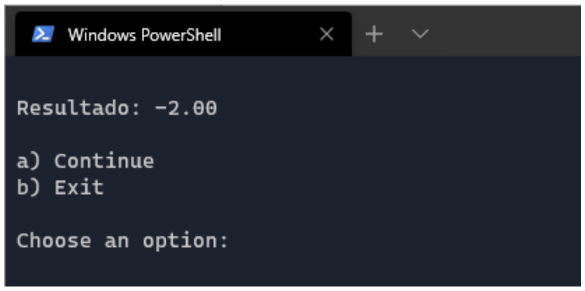
As pointed throughout the literature review the rule of Sarrus is used to calculate determinants of matrixes of multiple sizes; however, the algorithm of the MK75 program can only validate calculations of matrixes of size 2 or 3. Likewise, this part of the program can be accessed trough the main menu as option g).

Although the MK75 program will read matrixes of bigger sizes and mark those as valid data the determinant of those matrixes will not be calculated. Thus, printing "**Matrix size needs to be smaller or equal to 3"** in the system's terminal. All validations aside, the program will print the value of the determinant as normally intended.



```
Matrix size needs to be smaller or equal than 3

a) Continue
b) Exit

Choose an option:
```

Figure 16: When a matrix is to be calculated, the program will skip the action automatically if the current matrix size is not equal or lesser than 3.

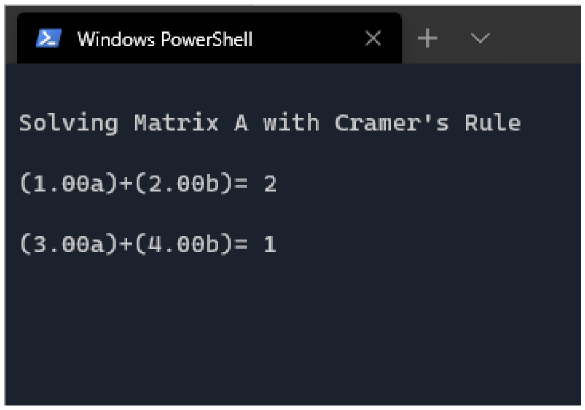


```
Resultado: -2.00

a) Continue
b) Exit

Choose an option:
```

Figure 17: The determinant is displayed whenever the current matrix size is optimal to perform a Sarrus operation.

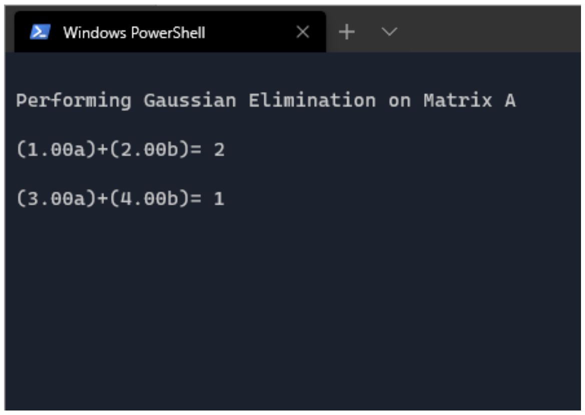
3.9 System of Equations: Vector of Constants

Although, one of the operations that hold upmost importance in linear algebra is solving systems of equations, there are two operations provided under the MK75 program that will solve a system of equations and therefore make use of the vector of constants, both of those operations use the same algorithm, otherwise said, the same part of the program. Not only whenever a system of equations is selected to be calculated the program will ask for a constant value, but also a corresponding equation will be formed dynamically according to the current matrix size and its entries.



```
Windows PowerShell

Solving Matrix A with Cramer's Rule

(1.00a)+(2.00b)= 2

(3.00a)+(4.00b)= 1
```

Figure 18: User prompted with equations as is asked to enter data for the Vector of Constants while solving with Cramer's rule.



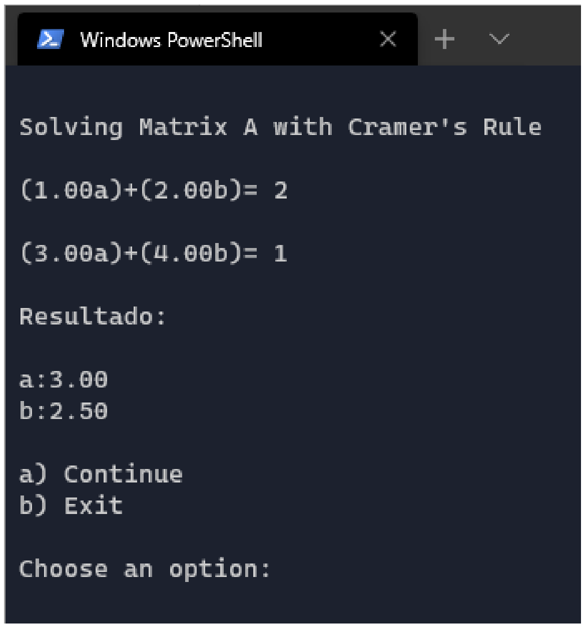
```
Windows PowerShell

Performing Gaussian Elimination on Matrix A

(1.00a)+(2.00b)= 2

(3.00a)+(4.00b)= 1
```

Figure 19: User prompted with equations as is asked to enter data for the Vector of Constants while solving by gaussian elimination.

3.10 System of Equations: Cramer's Rule.

The Cramer's rule is formally known as an alternative to gaussian elimination in linear algebra; however, it is not used for computer-based algorithms because of its intensity while computing determinants for each variable. Furthermore, because this algorithm is of asynchronous nature, as new technologies like multi-threading and quantum computing become optimal this operation could become a part of bigger algorithms.[10]

This part of the program is optimized for matrixes of smaller sizes, it performs all necessary calculations to a minimal extent thus being able to compute without running out of RAM and therefore solving a system of equations using Cramer's rule.



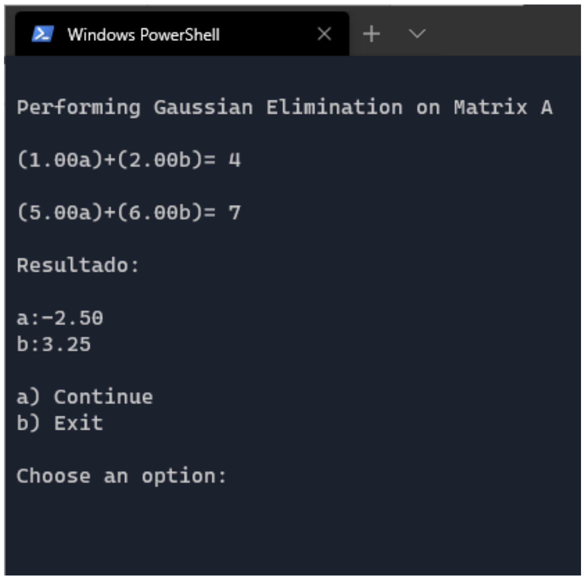
```
Windows PowerShell

Solving Matrix A with Cramer's Rule

(1.00a)+(2.00b)= 2

(3.00a)+(4.00b)= 1

Resultado:

a:3.00
b:2.50

a) Continue
b) Exit

Choose an option:
```

Figure 20: A system of equations is solved using Cramer's Rule.

3.11 System of Equations: Gaussian Elimination

Likewise, the algorithm of a gaussian elimination will also solve a system of equations. Although during the Cramer's algorithm the triangular form is used multiple times to calculate all the determinants, this algorithm will only make use of the triangular form once and then perform a backward substitution. Thus, performing better than the algorithm translated from Crammer's rule.

This extension of the program will solve any consistent system of equations with finite values. And stress less memory allocation when operating with bigger matrixes.

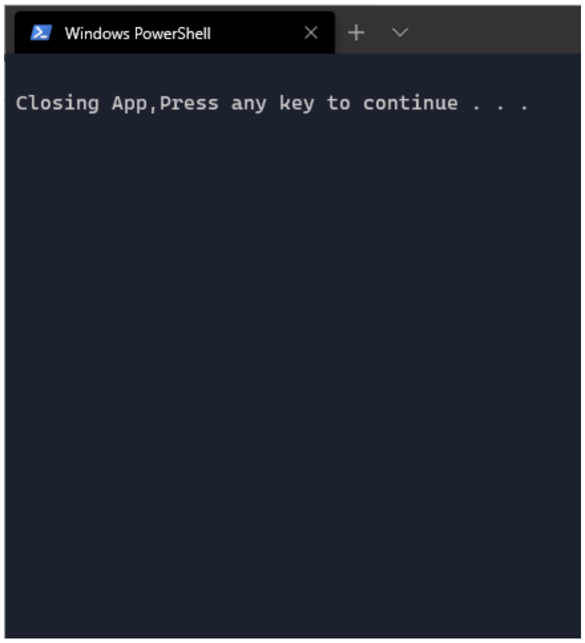


```
Performing Gaussian Elimination on Matrix A

(1.00a)+(2.00b)= 4

(5.00a)+(6.00b)= 7

Resultado:

a:-2.50
b:3.25

a) Continue
b) Exit

Choose an option:
```

Figure 21: A system of equations is solved using gaussian elimination.

3.12 End of Execution

Upon selecting the exit option from any of the MK75 menus, the execution of the program will be paused and shortly after a key is pressed the program's process will be terminated. To start the program again go to point 2.1 and execute the corresponding command.



```
Closing App,Press any key to continue . . .
```

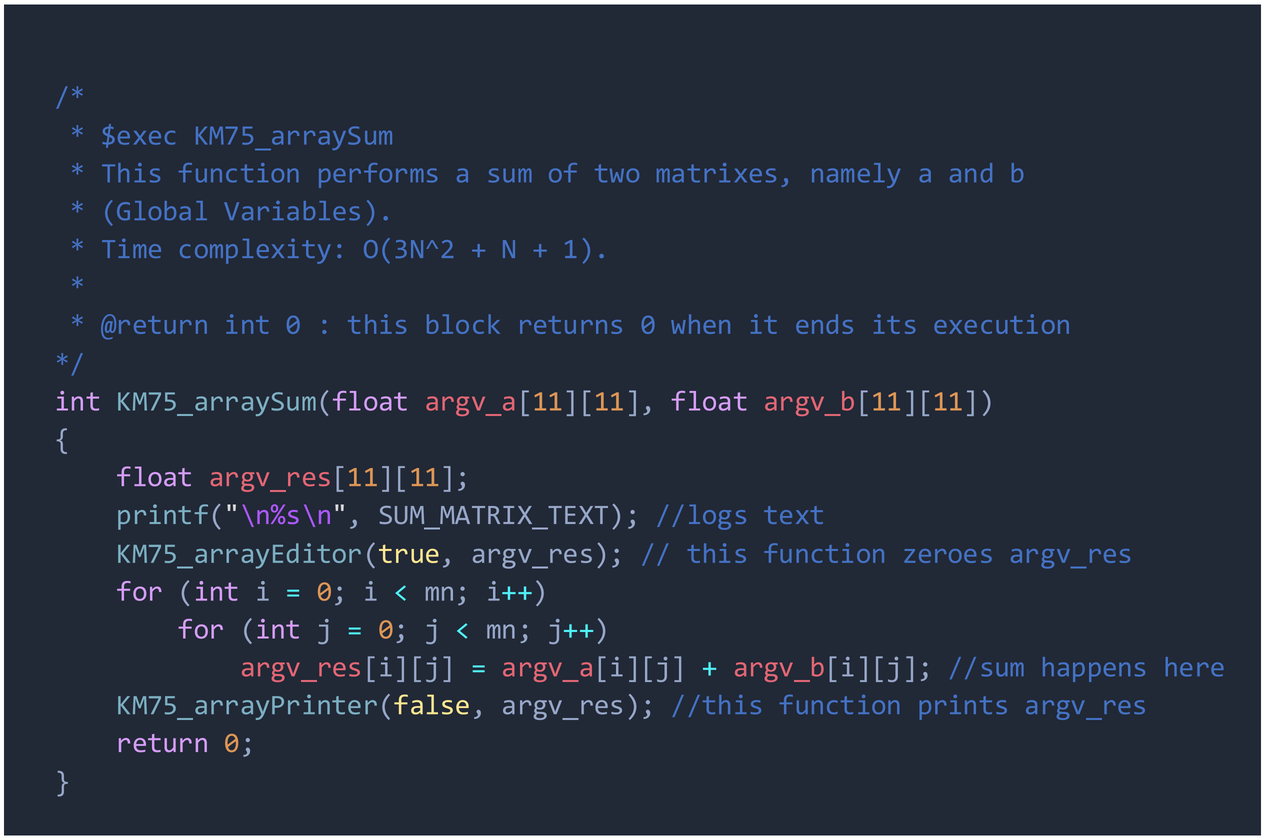
Figure 22: The user's terminal is instructed to close the program and waits for the user's input to end its execution, depending on what system the user is using the terminal will or will not close automatically.

4.1 Algorithms: Matrix Summation

The program uses a simple function, named KM75_arraySum, that receives data from two matrixes of type float: *argv_a*, and *argv_b*. and by using two loops can navigate through the indexes of a two dimensional vector, a matrix. The sum is calculated for each of the indexes and saved inside argv_res another matrix of type float and then printed in the system's terminal to display the result.

Although the time complexity is calculated for each function and represented in big O notation throughout the whole extent of the main program, it is completely irrelevant for this paper. But even so, it is still available to the reader through the code comments:

Ex. for this algorithm the time of execution is exactly $O(3N^2 + N + 1)$.

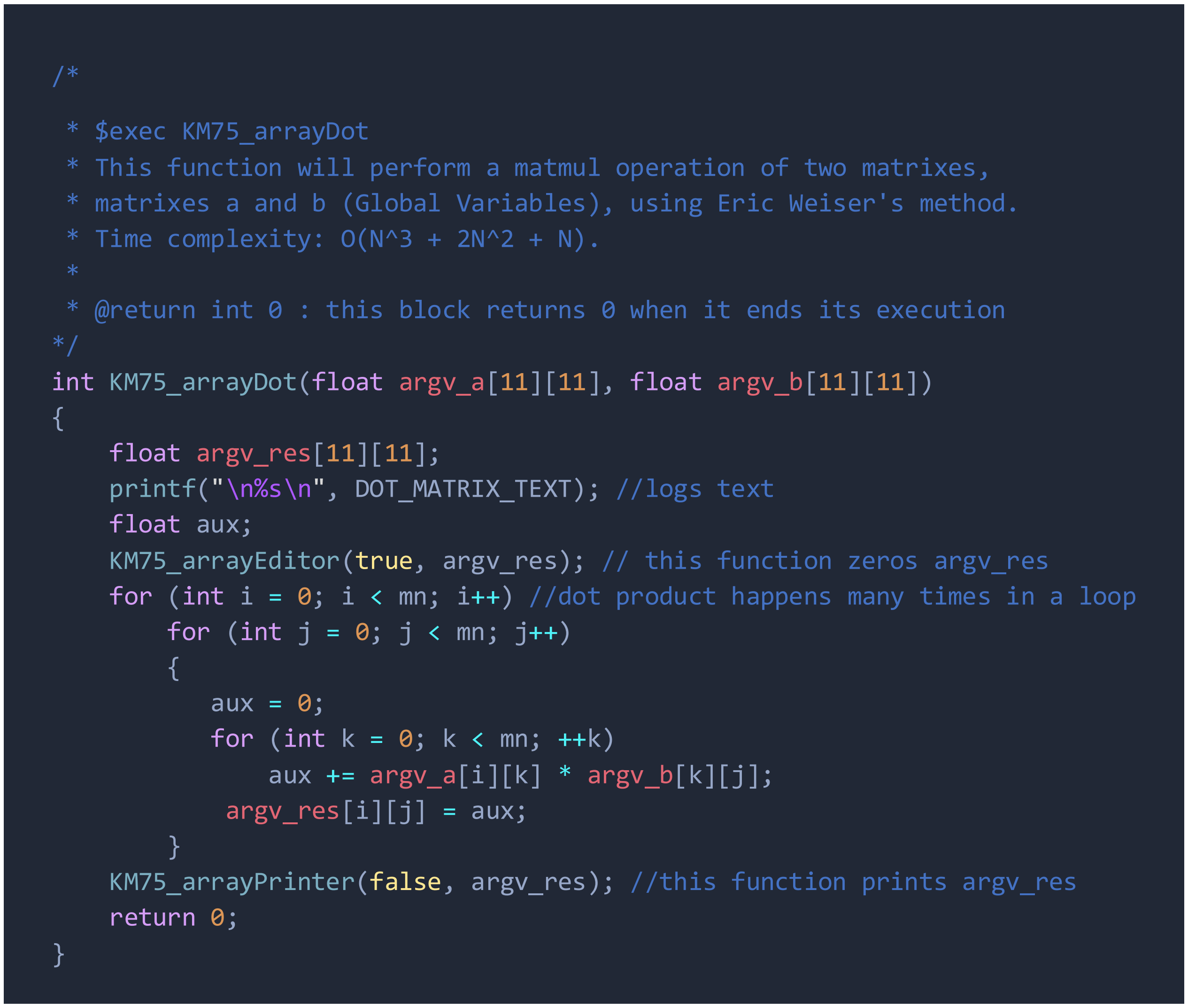


```c
/*
 * $exec KM75_arraySum
 * This function performs a sum of two matrixes, namely a and b
 * (Global Variables).
 * Time complexity: O(3N^2 + N + 1).
 *
 * @return int 0 : this block returns 0 when it ends its execution
*/
int KM75_arraySum(float argv_a[11][11], float argv_b[11][11])
{
    float argv_res[11][11];
    printf("\n%s\n", SUM_MATRIX_TEXT); //logs text
    KM75_arrayEditor(true, argv_res); // this function zeroes argv_res
    for (int i = 0; i < mn; i++)
        for (int j = 0; j < mn; j++)
            argv_res[i][j] = argv_a[i][j] + argv_b[i][j]; //sum happens here
    KM75_arrayPrinter(false, argv_res); //this function prints argv_res
    return 0;
}
```

Figure 23: Algorithm written in C to calculate the sum of two bidimensional vectors

4.2 Algorithms: Matrix Multiplication

KM75_arrayDot, just like the function before, is a function that receives data from two matrixes of type float: *argv_a*, and *argv_b*. and by using three loops the logic is able to navigate through the indexes of two two-dimensional vectors, two matrixes. And as stated before this algorithm performs a series of dot product operations to calculate each of the values of a product matrix *argv_res*, which after the calculated is also printed in the terminal in matrix form to display the result. Although matrixes A and B are used to perform the calculation, none of their values are being modified, therefore ready to execute and test other operations.

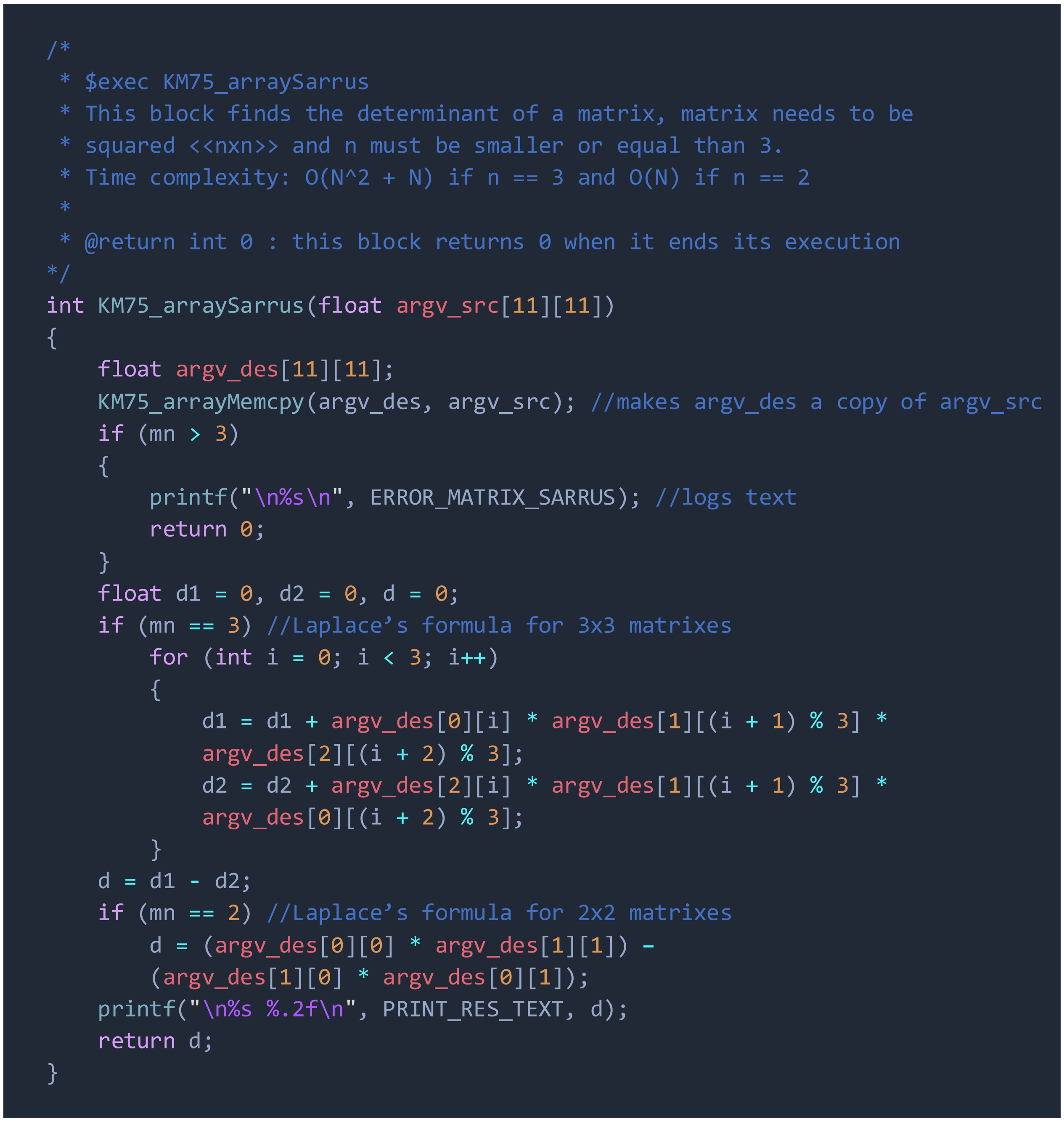


```c
/*
 * $exec KM75_arrayDot
 * This function will perform a matmul operation of two matrixes,
 * matrixes a and b (Global Variables), using Eric Weiser's method.
 * Time complexity: O(N^3 + 2N^2 + N).
 *
 * @return int 0 : this block returns 0 when it ends its execution
*/
int KM75_arrayDot(float argv_a[11][11], float argv_b[11][11])
{
    float argv_res[11][11];
    printf("\n%s\n", DOT_MATRIX_TEXT); //logs text
    float aux;
    KM75_arrayEditor(true, argv_res); // this function zeros argv_res
    for (int i = 0; i < mn; i++) //dot product happens many times in a loop
        for (int j = 0; j < mn; j++)
        {
           aux = 0;
           for (int k = 0; k < mn; ++k)
               aux += argv_a[i][k] * argv_b[k][j];
            argv_res[i][j] = aux;
        }
    KM75_arrayPrinter(false, argv_res); //this function prints argv_res
    return 0;
}
```

Figure 24: Algorithm written in C to calculate the multiplication of two bidimensional vectors

4.3 Algorithms: Matrix Determinant

This function is an algebraic equivalent to Laplace's formula applied to the Rule of Sarrus, which is also shortened by using a single loop and two Boolean operations *if*.

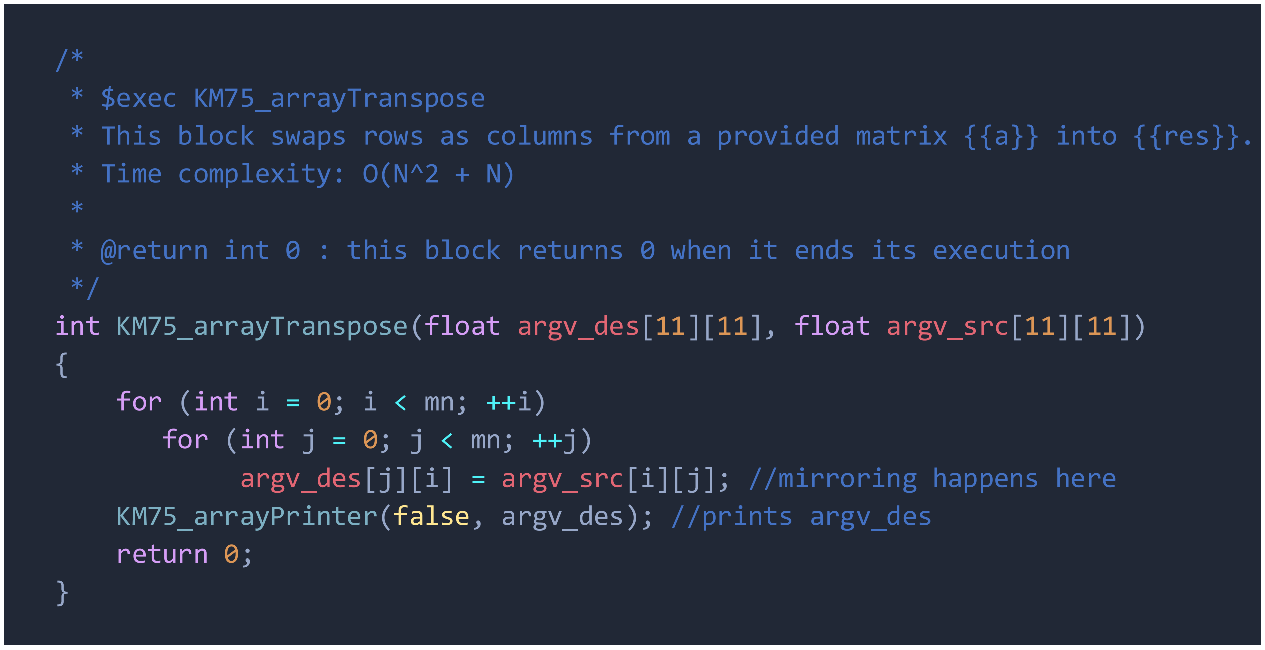


```c
/*
 * $exec KM75_arraySarrus
 * This block finds the determinant of a matrix, matrix needs to be
 * squared <<nxn>> and n must be smaller or equal than 3.
 * Time complexity: O(N^2 + N) if n == 3 and O(N) if n == 2
 *
 * @return int 0 : this block returns 0 when it ends its execution
*/
int KM75_arraySarrus(float argv_src[11][11])
{
    float argv_des[11][11];
    KM75_arrayMemcpy(argv_des, argv_src); //makes argv_des a copy of argv_src
    if (mn > 3)
    {
        printf("\n%s\n", ERROR_MATRIX_SARRUS); //logs text
        return 0;
    }
    float d1 = 0, d2 = 0, d = 0;
    if (mn == 3) //Laplace’s formula for 3x3 matrixes
        for (int i = 0; i < 3; i++)
        {
            d1 = d1 + argv_des[0][i] * argv_des[1][(i + 1) % 3] *
            argv_des[2][(i + 2) % 3];
            d2 = d2 + argv_des[2][i] * argv_des[1][(i + 1) % 3] *
            argv_des[0][(i + 2) % 3];
        }
    d = d1 - d2;
    if (mn == 2) //Laplace’s formula for 2x2 matrixes
        d = (argv_des[0][0] * argv_des[1][1]) -
        (argv_des[1][0] * argv_des[0][1]);
    printf("\n%s %.2f\n", PRINT_RES_TEXT, d);
    return d;
}
```

Figure 25: Algorithm written in C to calculate the determinant of a bidimensional vector

4.4 Algorithms: Transpose Matrix

Following the principles of a transpose operation, a provided matrix argv_src is iterated with two loops, one nested inside the other, at this moment a matrix argv_des mirrors the indexes from the provided matrix. After the transpose operation is concluded a resulting matrix is ready to print. All resulting arrays and vectors are printed using the program's utility to print arrays, this function will only be referenced in the document as a utility; however, the function or any other utility function will not be analyzed.

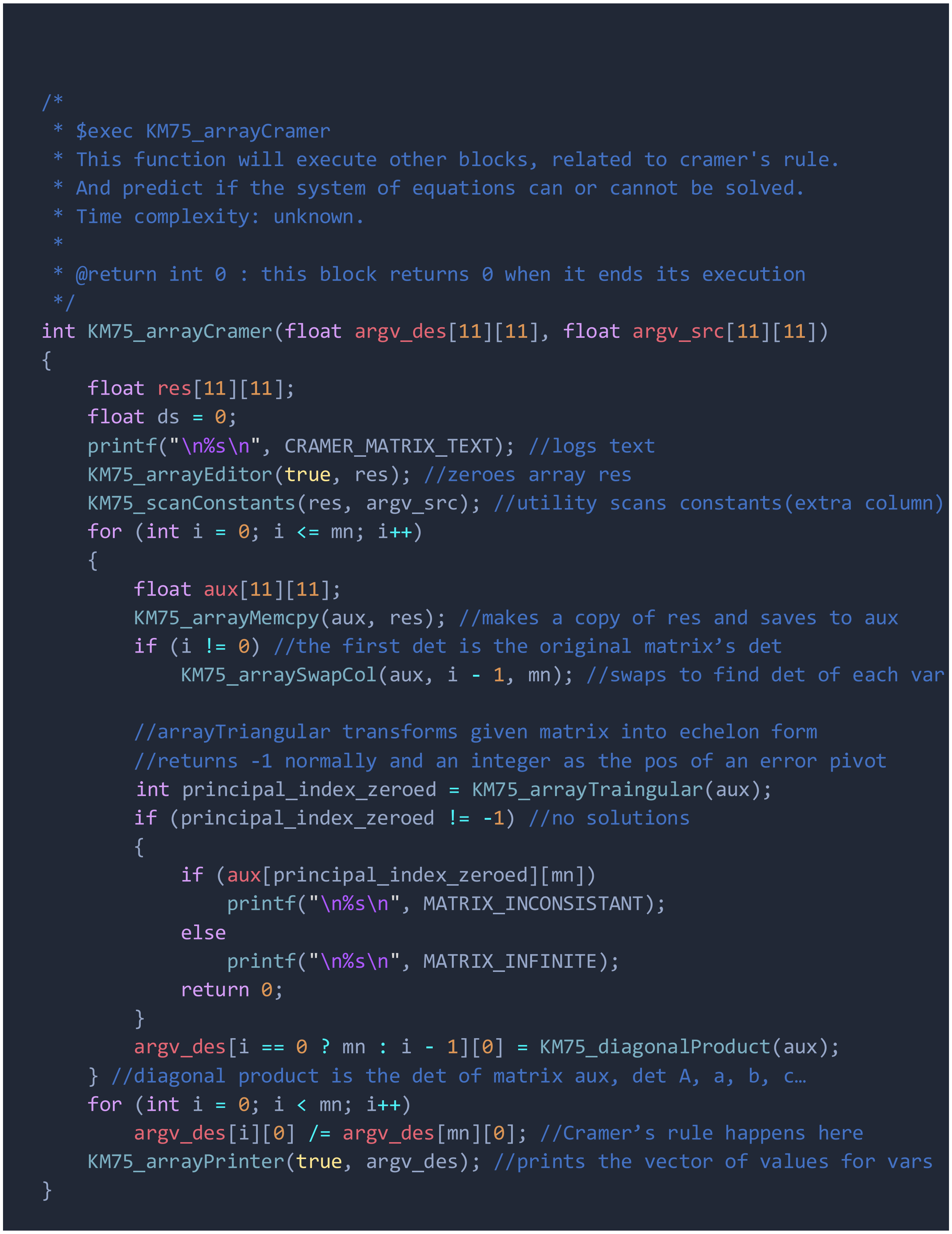


```c
/*
 * $exec KM75_arrayTranspose
 * This block swaps rows as columns from a provided matrix {{a}} into {{res}}.
 * Time complexity: O(N^2 + N)
 *
 * @return int 0 : this block returns 0 when it ends its execution
 */
int KM75_arrayTranspose(float argv_des[11][11], float argv_src[11][11])
{
    for (int i = 0; i < mn; ++i)
       for (int j = 0; j < mn; ++j)
            argv_des[j][i] = argv_src[i][j]; //mirroring happens here
    KM75_arrayPrinter(false, argv_des); //prints argv_des
    return 0;
}
```

Figure 26: Algorithm written in C to calculate the transpose of a bidimensional vector

4.5 Algorithms: Cramer's Rule

This algorithm is designed after the formula appointed as the Cramer's Rule, the formula shows that each vale of each variable is defined by determinant of a matrix of each variable divided by the determinant of the original matrix. (see Formula 4)

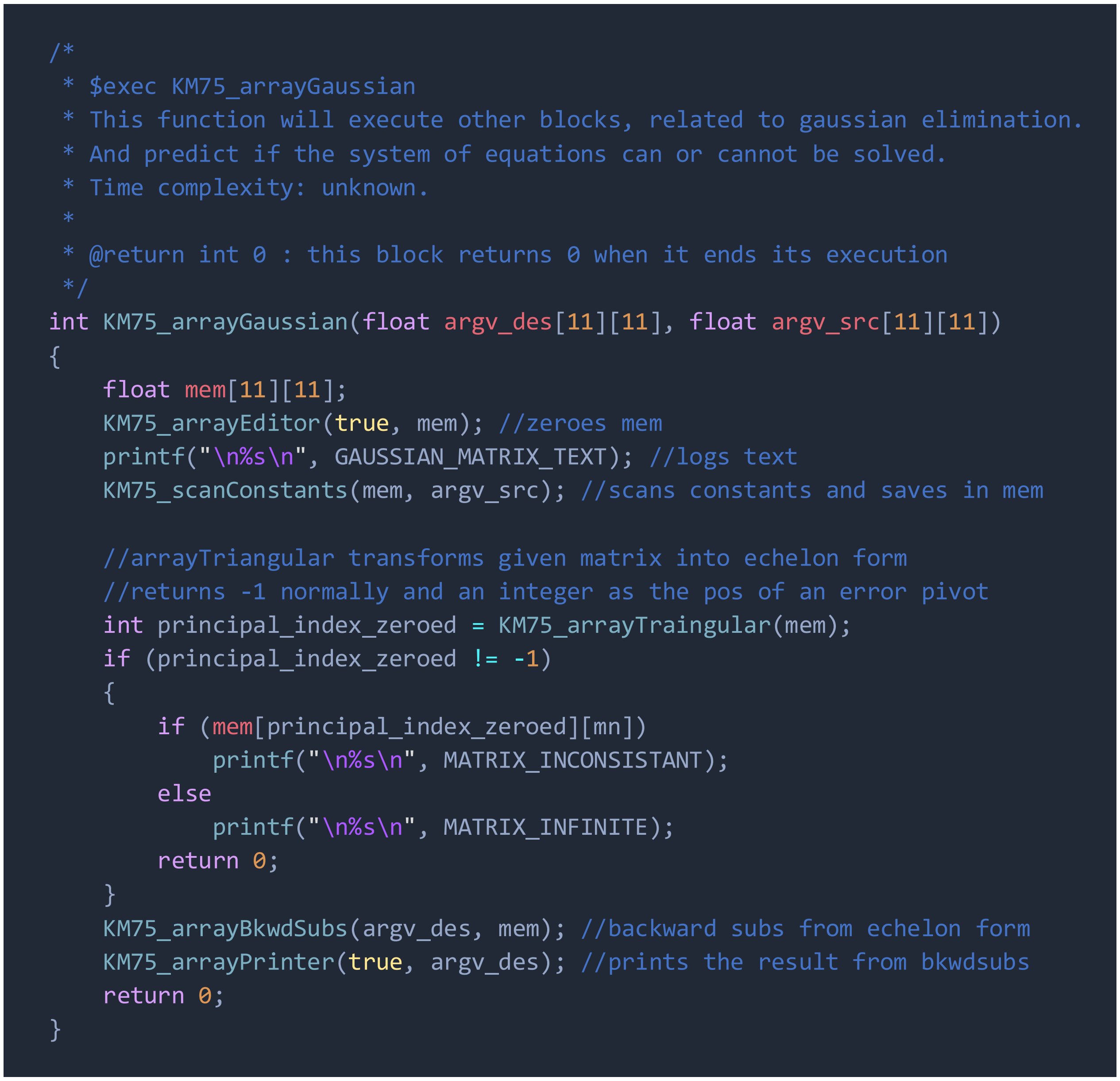


```c
/*
 * $exec KM75_arrayCramer
 * This function will execute other blocks, related to cramer's rule.
 * And predict if the system of equations can or cannot be solved.
 * Time complexity: unknown.
 *
 * @return int 0 : this block returns 0 when it ends its execution
 */
int KM75_arrayCramer(float argv_des[11][11], float argv_src[11][11])
{
    float res[11][11];
    float ds = 0;
    printf("\n%s\n", CRAMER_MATRIX_TEXT); //logs text
    KM75_arrayEditor(true, res); //zeroes array res
    KM75_scanConstants(res, argv_src); //utility scans constants(extra column)
    for (int i = 0; i <= mn; i++)
    {
        float aux[11][11];
        KM75_arrayMemcpy(aux, res); //makes a copy of res and saves to aux
        if (i != 0) //the first det is the original matrix’s det
            KM75_arraySwapCol(aux, i - 1, mn); //swaps to find det of each var

        //arrayTriangular transforms given matrix into echelon form
        //returns -1 normally and an integer as the pos of an error pivot
        int principal_index_zeroed = KM75_arrayTraingular(aux);
        if (principal_index_zeroed != -1) //no solutions
        {
            if (aux[principal_index_zeroed][mn])
                printf("\n%s\n", MATRIX_INCONSISTANT);
            else
                printf("\n%s\n", MATRIX_INFINITE);
            return 0;
        }
        argv_des[i == 0 ? mn : i - 1][0] = KM75_diagonalProduct(aux);
    } //diagonal product is the det of matrix aux, det A, a, b, c…
    for (int i = 0; i < mn; i++)
        argv_des[i][0] /= argv_des[mn][0]; //Cramer’s rule happens here
    KM75_arrayPrinter(true, argv_des); //prints the vector of values for vars
}
```

Figure 27: Algorithm written in C to solve a system of equations of a bidimensional vector and a vector of constants

4.3 Algorithms: Gaussian Elimination

The gaussian elimination algorithm executes a series of utility blocks starting by zeroing the result array, second scanning for the vector of constants using the source matrix *argv_src*, third using the triangular array function to transform the given matrix into the echelon form, finally performing backward substitution and saving the values into the *argv_des* vector. Thus, having position to the solution of the system of equations. Once the algorithm is concluded the program prints the resulting vector as the solution of the system of equations.



```c
/*
 * $exec KM75_arrayGaussian
 * This function will execute other blocks, related to gaussian elimination.
 * And predict if the system of equations can or cannot be solved.
 * Time complexity: unknown.
 *
 * @return int 0 : this block returns 0 when it ends its execution
 */
int KM75_arrayGaussian(float argv_des[11][11], float argv_src[11][11])
{
    float mem[11][11];
    KM75_arrayEditor(true, mem); //zeroes mem
    printf("\n%s\n", GAUSSIAN_MATRIX_TEXT); //logs text
    KM75_scanConstants(mem, argv_src); //scans constants and saves in mem

    //arrayTriangular transforms given matrix into echelon form
    //returns -1 normally and an integer as the pos of an error pivot
    int principal_index_zeroed = KM75_arrayTraingular(mem);
    if (principal_index_zeroed != -1)
    {
        if (mem[principal_index_zeroed][mn])
            printf("\n%s\n", MATRIX_INCONSISTANT);
        else
            printf("\n%s\n", MATRIX_INFINITE);
        return 0;
    }
    KM75_arrayBkwdSubs(argv_des, mem); //backward subs from echelon form
    KM75_arrayPrinter(true, argv_des); //prints the result from bkwdsubs
    return 0;
}
```

Figure 28: Algorithm written in C to solve a system of equations of a bidimensional vector and a vector of constants

5.1 References

[1] Navdeep Singh, "Good Code vs Bad Code - Better Programming," *Medium*, Jul. 25, 2017. https://medium.com/better-programming/good-code-vs-bad-code-35624b4e91bc (accessed Jun. 21, 2020).

[2] Benjamin C. Pierce, *Types and Programming Languages*. Pennsylvania: University of Pennsylvania, 2002.

[3] G. E. Tseitlin, "Algebraic Algorithmics: Theory and Applications," *Cybernetics and Systems Analysis*, vol. 39, no. 1, pp. 6–15, Jan. 2003, Accessed: Jun. 21, 2020. [Online].

[4] Jane Day and Brian Peterson, "Growth in Gaussian Elimination on JSTOR," *The American Mathematical Monthly*, Jun. 1988. https://www.jstor.org/stable/2322755?origin=crossref&seq=1 (accessed Jun. 21, 2020).

[5] G. Cramer, *Introduction à l'Analyse des lignes Courbes algébriques*. Geneva: Europeana, 1750.

[6] G. Fischer, *Analytische Geometrie*, 4th ed. Wiesbaden: Vieweg, 1985.

[7] Khan Academy, "Properties of matrix multiplication," *Khan Academy Organization*, Jan. 01, 2020. https://www.khanacademy.org/math/precalculus/x9e81a4f98389efdf:matrices/x9e81a4f98389efdf:properties-of-matrix-multiplication/a/properties-of-matrix-multiplication (accessed Jun. 24, 2020).

[8] Randerson, "How to Run C-Program in Command Prompt," *Medium*, Feb. 16, 2018. https://medium.com/@randerson112358/how-to-run-c-program-in-command-prompt-e435186cd162 (accessed Jun. 22, 2020).

[9] T. A. Whitelaw, *Introduction to Linear Algebra, 2nd edition*. CRC Press, 1991.

[10] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum algorithm for linear systems of equations," *Physical Review Letters*, vol. 103, no. 15, p. 150502, Oct. 2009, doi: 10.1103/PhysRevLett.103.150502.