Matrix Inversion Method Using C++ and Python

[1]Oshinubi, K. [2]Ikpeme, E.J., [3]Daramola, F., [4]Afolabi, A.O., [5]Nwaocha, O. V. & [6]Longe, O.B.
[1]Department of Informatics & Mathematics, University of Grenoble, Grenoble, France
[2&3]Dept. of Computer Science, Caleb University – Lagos
[4]Dept of Math and Science, Gambian-Lebanese International School – The Gambia
Department of Computer Science, National Open University of Nigeria, Abuja, Nigeria
Department of Information Systems, American University of Nigeria, Yola, Nigeria
E-mails: oshinubik@gmail.com, ikpemeelijahjames@outlook.com, fundaramola2013@gmail.com, tundejoelafolabi@gmail.com; onwaocha@noun.edu.ng, Olumide.longe@aun.edu.ng

ABSTRACT

Matrix inversion is a numerical method for solving a series of simultaneous equations. This term paper deals with using programming languages C++ and python to develop programs that can solve simultaneous equations with 2,3 or 4 variables (i.e. programs which can compute the values for 4x4, 3x3 and 3x3 matrices problems). In this paper, key factors in matrices such as determinants, adjugate, cofactors, transposes and determinants were reiterated for basic understanding. Going forward, it also briefly explains the benefit of the application of matrix inversion method to some real-life situations as well as the steps in solving the modelled problems using C++ and python.

Keywords – Matrix, C++, Python, Inversion, Modelling, Application.

iSTEAMS Multidisciplinary Conference Proceedings Reference Format
Oshinubi, K., Ikpeme E. James, Daramola, F., Afolabi, A. O, Nwaocha, O.V & Longe, O.B. (2019): Global Leadership and Implications for Organizations. Proceedings of the 20th iSTEAMS Multidisciplinary Trans-Atlantic Conference, KEAN University, New Jersey, United States of America. 10th – 12th October, 2019. Pp 9-42 www.isteams.net/usa2019 - DOI Affix - https://doi.org/ 10.22624/AIMS/iSTEAMS-2019/V20N1P2

1. INTRODUCTION

Matrix inversion is a numerical method developed for solving a series of simultaneous equations, this one of the areas in mathematics where matrix can be applied in solving simultaneous equation. Solving a series of equations with matrix inversion method requires for the series of equations to be converted to matrix form. From the equations we are to derive three matrices a square matrix called 'A', a colon matrix called 'B' and a colon matrix 'X' containing the variables to be found. An example below shows how this is done.

$$X + 2y = 4$$

$$3x - 5y = 1$$

From this equation we will use the coefficient of the equation to form our square matrix. A is called the coefficient of the matrix

$$A = \begin{pmatrix} 1 & 2 \\ 3 & -5 \end{pmatrix}$$

To get our 'B' matrix we use the numbers on left hand side of equation to obtain our B matrix
B = $\begin{pmatrix} 4 \\ 1 \end{pmatrix}$

The values in our X matrix will contain the variables to be found.

X = $\begin{pmatrix} X \\ Y \end{pmatrix}$

Using these matrices, we shall rewrite the series of simultaneous equations in matrix form.

This will give us the equation ***AX=B … (equation 1)***

$$\begin{pmatrix} 1 & 2 \\ 3 & -5 \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \end{pmatrix}$$

our aim is to get the values of the variables in matrix X, so if we divide equation 1 by A we get

X = A^{-1}B … (equation 2)

The equation gives us a method for solving the simultaneous equations.

In order to obtain the variables in X we must find the inverse of matrix A (A^{-1}) and multiple the result with matrix B to derive the values in matrix X.

The formula of inverse of a matrix is $\mathbf{A^{-1}} = \frac{1}{Det(A)} adj(A)$

Det(A) is the determinant of Matrix A

adj(A) is the adjugate of matrix A

1.1 Determinant
Determinant is a value that can be computed from a square matrix, the determinant of a Matrix A is denoted by Det(A) or |A|. since this term paper only covers 2x2,3x3 and 4x4 matrices, we shall only look at getting the determinant of these square matrices.

- **2x2 matrix determinant**

Given a matrix **A =** $\begin{vmatrix} a & b \\ c & d \end{vmatrix}$

|A| = ***ad – bc***

- **3x3 matrix determinant**

Given a matrix A = $\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$

$$|A| = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

$$|A| = a(ei-hf) - b(di-fg) + c(dh-ge)$$

- **4x4 matrix determinant**
-

Given a matrix A = $\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$

$$|A| = a\begin{bmatrix} f & g & h \\ j & k & l \\ n & o & p \end{bmatrix} - b\begin{bmatrix} e & g & h \\ i & k & l \\ m & o & p \end{bmatrix} + c\begin{bmatrix} e & f & h \\ i & j & l \\ m & n & p \end{bmatrix} - d\begin{bmatrix} e & f & g \\ i & j & k \\ m & n & o \end{bmatrix}$$

$$|A| = a[f(kp-ol)-g(jp-ln)+h(jo-kn)] - b[e(kp-lo)-g(ip-lm)+h(io-km)] + c[e(jo-kn)-f(io-km)+g(in-jm)]$$

1.2 Adjugate Of Matrix

The adjugate matrix of a square matrix is the transpose of it cofactor matrix. $Adj(A) = C^T$ where C is the cofactor matrix of A.

1.2.1 COFACTOR

A minor of a matrix **A** is the determinant of some smaller matrix, cut down from A by removing one or more columns or rows. Minors are required for calculating cofactors, which are useful for computing determinant and inverse of a square matrix.

If A is a square matrix, then the minor of the entry in *i*th row and *j*th column is the determinant of the submatrix formed by deleting the i-th row and j-th column. This number is often denoted by $M_{i,j}$ .The cofactor is obtained by multiplying the minor by C_{ij} $(-1)^{i+j}$. consider this 3 x 3 matrix,

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 2 & 2 \end{bmatrix}$$

To find the minor M2,3 and the cofactor C2,3 we find the determinant of the above matrix with row 2 and column 3 removed.

$$M_{2,3} = \det\begin{bmatrix} 1 & 2 & \\ & & \\ 3 & 2 & \end{bmatrix} = \det\begin{vmatrix} 1 & 2 \\ -1 & 9 \end{vmatrix} = (9-(-2)) = 11$$

The cofactor of the entry (2,3) is

$$C_{2,3} = (-1)^{2+3}(M_{2,3}) = -11 \text{ -------(equation c)}$$

Let **A** be an *m* × *n* matrix and *k* an integer with 0 < *k* ≤ *m*, and *k* ≤ *n*. A ***k* × *k* minor** of **A**, also called **minor determinant of order k** of **A** or, if m = n, **(n-k):th minor determinant** of **A**, with the word "determinant" often omitted and the word "order" sometimes replaced by "degree", is the determinant of a *k* × *k* matrix obtained from **A** by deleting *m* − *k* rows and *n* − *k* columns. Sometimes the term is used to refer to the *k* × *k* matrix obtained from **A** as above (by deleting *m* − *k* rows and *n* − *k* columns), but this matrix should be referred to as a **(square) submatrix** of **A**, leaving the term "minor" to refer to the determinant of this matrix.

$$\mathbf{C} = \begin{bmatrix} \boldsymbol{C11} & \boldsymbol{C12} & \cdots & \boldsymbol{C1n} \\ \boldsymbol{C21} & \boldsymbol{C22} & \ddots & \boldsymbol{C2n} \\ \vdots & \vdots & \cdots & \vdots \\ \boldsymbol{Cn1} & \boldsymbol{Cn2} & & \boldsymbol{Cnn} \end{bmatrix}$$

Using the method, we used to find C2,3 in equation c we can apply this rule to find and cofactor that's $\mathbf{C_{ij}}$

1.3 Transpose

In linear algebra, the transpose of a matrix is an operator which flips a matrix over it's diagonal, that us switch the row and column indices of the matrix by producing a new matrix $\mathbf{A^T}$.

It is achieved by any one of the following methods

- Write the columns of A as the rows of $\mathbf{A^T}$
- Write the rows of A as the columns of $\mathbf{A^T}$
- Mirror A over its primary diagonal to obtain $\mathbf{A^T}$

$$\mathbf{A} = \begin{vmatrix} \boldsymbol{a} & \boldsymbol{b} \\ \boldsymbol{c} & \boldsymbol{d} \end{vmatrix}$$

$$\mathbf{A^T} = \begin{vmatrix} \boldsymbol{a} & \boldsymbol{c} \\ \boldsymbol{b} & \boldsymbol{d} \end{vmatrix}$$

1.4 Inverse Matrix

Inverse matrix formula is $\boldsymbol{A^{-1}} = \frac{1}{\boldsymbol{Det(A)}}\,\boldsymbol{adj(A)}$

Examples

Example 1
2x – 3y = 5
3x + 4y = 3

A = $\begin{vmatrix} 2 & -3 \\ 3 & 4 \end{vmatrix}$

X = $\begin{vmatrix} x \\ y \end{vmatrix}$

B = $\begin{vmatrix} 5 \\ 3 \end{vmatrix}$

AX=B

X = A^{-1}B

A^{-1}= $\frac{1}{Det(A)} adj(A)$

Det(A) = (2*4) – (-3*3)

Det(A) = 8 + 9 = 17

Adj(A) = $\begin{vmatrix} 4 & 3 \\ -3 & 2 \end{vmatrix}$

A^{-1} = $\frac{1}{17}\begin{vmatrix} 4 & 3 \\ -3 & 2 \end{vmatrix}$

A^{-1} = $\begin{vmatrix} \frac{4}{17} & \frac{3}{17} \\ -\frac{3}{17} & \frac{2}{17} \end{vmatrix}$

X = A^{-1}B

X = $\begin{vmatrix} \frac{4}{17} & \frac{3}{17} \\ -\frac{3}{17} & \frac{2}{17} \end{vmatrix} * \begin{vmatrix} 5 \\ 3 \end{vmatrix}$

X = $\begin{vmatrix} \frac{20}{17} + \frac{9}{17} \\ -\frac{15}{17} + \frac{6}{17} \end{vmatrix} = \begin{vmatrix} \frac{29}{17} \\ -\frac{9}{17} \end{vmatrix}$

X = $\frac{29}{17}$ **and y =** $-\frac{9}{17}$

Example 2

x + 2y + 3z = 2

2x + 3y +4z = -2

3x + 2y + 2z = 3

$$\boldsymbol{A} = \begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} \\ \mathbf{2} & \mathbf{3} & \mathbf{4} \\ \mathbf{3} & \mathbf{2} & \mathbf{2} \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{2} \\ \mathbf{-2} \\ \mathbf{3} \end{bmatrix}$$

$$\mathbf{Det(A)} = 1\begin{vmatrix} 3 & 4 \\ 2 & 2 \end{vmatrix} - 2\begin{vmatrix} 2 & 4 \\ 3 & 2 \end{vmatrix} + 3\begin{vmatrix} 2 & 3 \\ 3 & 2 \end{vmatrix}$$

Det(A) =1(6-8) -2(4-12) + 3(4-9)

Det(A) = -2 – 2(-8) +3(-5)

Det(A) = -2+16-15

Det(A) = -1

$$\mathbf{A^C} = \begin{bmatrix} \boldsymbol{C11} & \boldsymbol{-C12} & \boldsymbol{C13} \\ \boldsymbol{-C21} & \boldsymbol{C22} & \boldsymbol{-C23} \\ \boldsymbol{C31} & \boldsymbol{-C32} & \boldsymbol{C33} \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} \\ \mathbf{2} & \mathbf{3} & \mathbf{4} \\ \mathbf{3} & \mathbf{2} & \mathbf{2} \end{bmatrix}$$

C11 = 6-8 = -2

C12 = – (4-12) = -(-8) = 8

C13 = 4-9 = -5

C21 = -(4-6) = -(-2) = 2

C22 = 2-9 = -7

C23 = -(2-6) = -(-4) = 4

C31 = (8-9) = -1

C32 = -(4-6) = -(-2) = 2

C33 = 3-4 = -1

$$A^C = \begin{bmatrix} -2 & 8 & -5 \\ 2 & -7 & 4 \\ -1 & 2 & -1 \end{bmatrix}$$

$A^T = [A^c]^T$

-1

$$A^T = \begin{bmatrix} -2 & 2 & -1 \\ 8 & -7 & 2 \\ -5 & 4 & -1 \end{bmatrix}$$

$$A^{-1} = \frac{1}{Det(A)} adj(A)$$

$$A^{-1} = -1\begin{bmatrix} -2 & 2 & -1 \\ 8 & -7 & 2 \\ -5 & 4 & -1 \end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} 2 & -2 & 1 \\ -8 & 7 & -2 \\ 5 & -4 & 1 \end{bmatrix}$$

$$X = A^{-1}B$$

$$X = \begin{bmatrix} 2 & -2 & 1 \\ -8 & 7 & -2 \\ 5 & -4 & 1 \end{bmatrix}\begin{bmatrix} 2 \\ -2 \\ 3 \end{bmatrix}$$

$$X = \begin{bmatrix} 4+4+3 \\ -16-14-6 \\ 10+8+3 \end{bmatrix}$$

$$X = \begin{bmatrix} 11 \\ -36 \\ 21 \end{bmatrix}$$

X = 11, y = -36, z = 21

2. APPLICATION OF MATRIX INVERSION IN DAILY LIFE

While matrices are used more in daily life than one would have thought, the inversion method of solving matrix problem is very much relevant in coding, computer aided designs, cryptography and so on.In physics related applications, matrices are used in the study of electrical circuits, quantum mechanics and optics. Engineers use matrix inversion to model physical systems and perform accurate calculations needed for complex mechanics to work.

Electronics networks, airplane and spacecraft, and in chemical engineering all require perfectly calibrated computations which are obtained from matrix transformations.Whereas in programming, matrices and inverse matrices are used for coding and encrypting messages. A message is made as a sequence of numbers in a binary format for communication and it follows code theory for solving. Graphic software such as Adobe Photoshop on your personal computer uses matrix inversion to process linear transformations to render images. A square matrix can represent a linear transformation of a geometric object.

For example, in the Cartesian X-Y plane, the matrix $\begin{matrix} 0 & -1 \\ 1 & 0 \end{matrix}$ reflects an object in the vertical Y axis. In a video game, this would render the upside-down mirror image of an assassin reflected in a pond of blood. If the video game has curved reflecting surfaces, such as a shiny metal shield, the matrix would be more complicated, to stretch or shrink the reflection. In robotics and automation, matrices are the basic components for the robot movements. The inputs for controlling robots are obtained based on the solutions derived from modelled simultaneous equations and using matrices inversion to solve such. These solutions produce very accurate movements.

3. USING C++ TO SOLVE A SERIES OF SIMULTANEOUS EQUATIONS BY MATRIX INVERSION METHOD

C++ is a general-purpose object-oriented programming language developed by Bjarne Stroustrup, and is an extension of the C language. C++ is considered a hybrid language because it encapsulates both low and high level language features. The program was written in C++ to solve a series of simultaneous equations using matrix inversion, it uses Object Oriented Programming concept.

A class was written that defines the methods and data members for creating matrix objects which have function to derive adjugate matrix, inverse matrix and find the variables of the simultaneous equation.

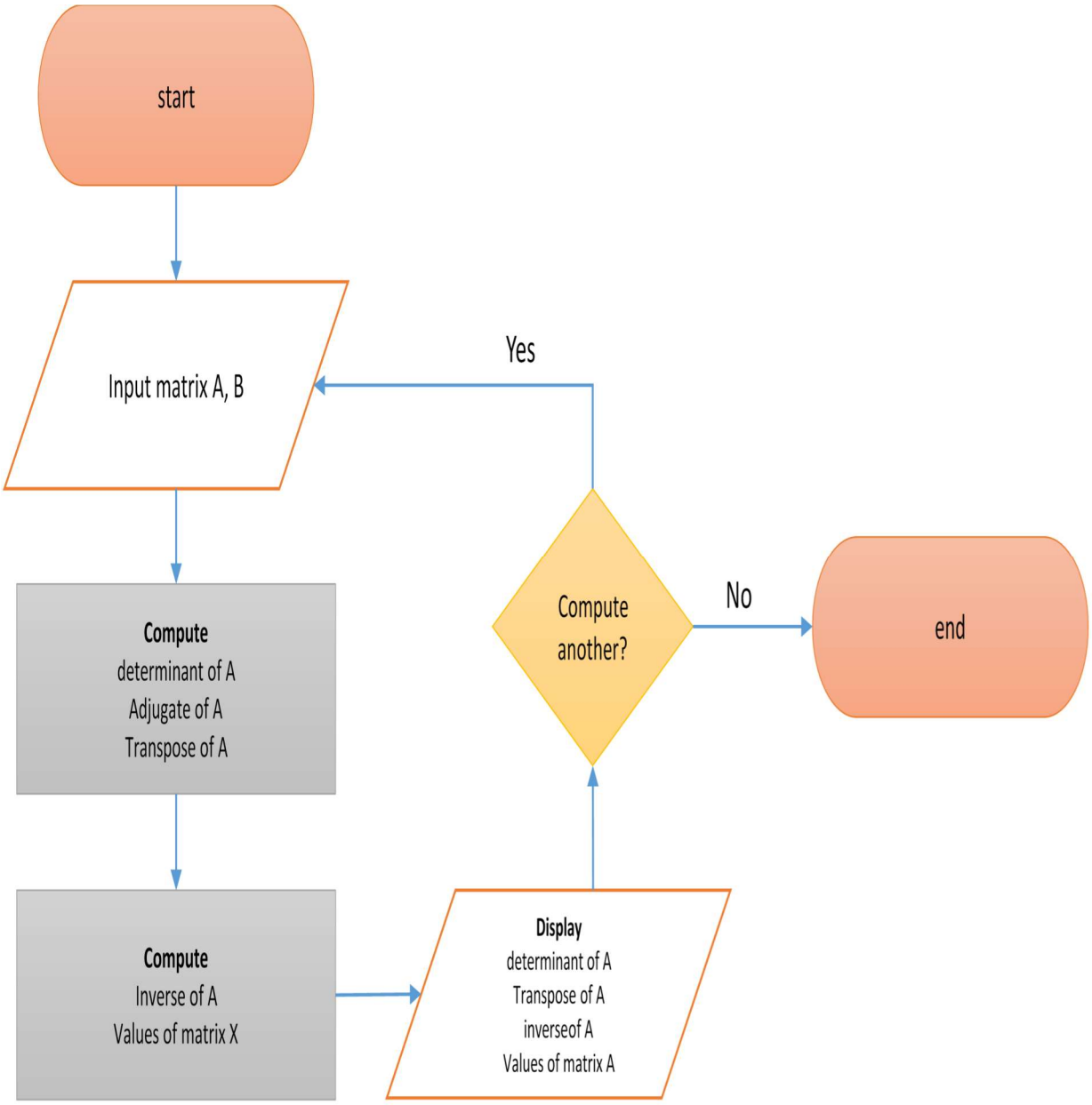

Fig 1: Program Flowchart

Class Definition code (Matrix.h file)

```
#pragma once

class Matrix
{
public:
        Matrix(int n,int **matrix,int* matrix_b);
        ~Matrix();
        int deteminant();
        int** adjugate();
        double** inverseMatrix();
        int** transpose();
        void displayMatrix();
        void displayBmatrix();
        void displayAdjugate();
        void displayInverse();
        void displayTranspose();
        void displayAnswer();
        void ComputeAnswer();
private:
        Matrix();
        int deteminant2x2();
        int deteminant3x3();
        int deteminant4x4();
        int deteminant4Handler(int a[3][3]);
        void adjugate2x2();
        void adjugate3x3();
        void adjugate4x4();
        void transpose2x2();
        void transpose3x3();
        void transpose4x4();
        int **A;
        int *B;
        int **Adjugate_Matrix;
        int **A_transpose;
        double* result;
        double **A_inverse;
        int sizeOfMatrix;
};
```

The class contains functions for finding determinants for 2x2 ,3x3,4x4 matrices. It determines which function to call based on the values proved to the constructor when a new matrix object is created. We hide the default construction because we need the user to specific the matrix type when creating an object. The construction created in the public section requires the matrix size, a 2D array of integer (2x2,3x3 or 4x4) and an array representing matrix B. It uses this values to initialize the data members of the object. The main functions of the class are determinant(), adjugate(), inverseMatrix(), transpose() and ComputeAnswer().

The **determinant** functions use the 'A' matrix size is used to call the right sub function to compute the determinant, if the size of the matrix is 2 the function uses determinant2x2(), else if the size is 3 the function uses determinant3x3(), else if it is 4 the function uses determinant4x4(). This sub functions are called depending on the size of the A matrix. Only one function is selected.

The **Adjugate** function uses the size of the 'A' matrix to determine which right sub function to call. If the size is 2 it calls adjugate2x2 sub function, else if the size is 3 it calls adjugate3x3 sub function, else if the size is 4 it calls adjugate4x4 sub function. Each of this sub functions have specific codes for computing for sizes 2 to 4.
The **transpose** function computes the transpose of the adjugate matrix, it also uses the size of Matrix A to call sub functions for specific sizes. The **Inverse** function computes the matrix inverse by multiplying the Transpose matrix by 1/det to generate A^{-1}.

The **computeAnswer** function using the formula $X = A^{-1}B$ to find the variables of the X matrix.
The constructor function performs all the computations once an object is created by call the following function in this order.

- determinant()
- Adjugate()
- Transpose()
- Inverse()
- computeAnswer()

The class also has methods for displaying determinant, adjugate matrix, transpose, inverse and values of variables in matrix X. this functions are used in main.cpp to generate output. This program was written without external libraries.

Class Implementation code (Matrix.cpp file)

```
#include"Matrix.h"
#include<iostream>
Matrix::Matrix(int n,int** matrix,int* matrix_B)
{
	if (n < 2)
	{
		n = 2;
	}
	else if (n > 4)
	{
		n = 4;
	}
	sizeOfMatrix = n;
	result = new double[sizeOfMatrix];
	B = new int[sizeOfMatrix];
	A = new int*[sizeOfMatrix];
	Adjugate_Matrix = new int*[sizeOfMatrix];
	A_inverse = new double*[sizeOfMatrix];
	A_transpose = new int*[sizeOfMatrix];
	for (int i = 0; i < sizeOfMatrix; i++)
```

```
	{
		A[i] = new int[sizeOfMatrix];
		Adjugate_Matrix[i] = new int[sizeOfMatrix];
		A_inverse[i] = new double[sizeOfMatrix];
		A_transpose[i] = new int[sizeOfMatrix];
	}
	for (int i = 0; i < n; i++)
	{
		for (int j = 0; j < n; j++)
		{
			A[i][j] = matrix[i][j];
		}
	}
	for (int i = 0; i < n; i++)
	{
		B[i] = matrix_B[i];
	}
	deteminant();
	adjugate();
	transpose();
	inverseMatrix();
	ComputeAnswer();
}
Matrix::~Matrix()
{
	//clear up used memory
	for (int i = 0; i < sizeOfMatrix; i++)
	{
		delete A[i];
	}
	delete A;
	delete B;
	for (int i = 0; i < sizeOfMatrix; i++)
	{
		delete Adjugate_Matrix[i];
	}
	delete Adjugate_Matrix;
	for (int i = 0; i < sizeOfMatrix; i++)
	{
		delete A_transpose[i];
	}
	delete A_transpose;
	delete result;
	for (int i = 0; i < sizeOfMatrix; i++)
```

```
		{
			delete A_inverse[i];
		}
		delete A_inverse;


}
int Matrix::deteminant()
{
		int det = 0;
		if (sizeOfMatrix == 2)
		{
			det = deteminant2x2();
		}
		else if(sizeOfMatrix == 3)
		{
			det =  deteminant3x3();
		}
		else if (sizeOfMatrix == 4)
		{
			det =  deteminant4x4();
		}
		return det;
}
int Matrix::deteminant2x2()
{
		if (sizeOfMatrix != 2)return 0;
		int det = (A[0][0] * A[1][1]) - (A[0][1] * A[1][0]);
		return det;
}
int Matrix::deteminant3x3()
{
		int col1 = A[0][0] * ((A[1][1] * A[2][2]) - (A[2][1] * A[1][2]));
		int col2 = -A[0][1] * ((A[1][0] * A[2][2]) - (A[2][0] * A[1][2]));
		int col3 = A[0][2] * ((A[1][0] * A[2][1]) - (A[2][0] * A[1][1]));
		return (col1 + col2 + col3);
}
int Matrix::deteminant4Handler(int a[3][3])
{
		int col1 = a[0][0] * ((a[1][1] * a[2][2]) - (a[2][1] * a[1][2]));
		int col2 = -a[0][1] * ((a[1][0] * a[2][2]) - (a[2][0] * a[1][2]));
		int col3 = a[0][2] * ((a[1][0] * a[2][1]) - (a[2][0] * a[1][1]));
		return (col1 + col2 + col3);
}
```

```
int Matrix::deteminant4x4()
{
	int A1[3][3];
	int A2[3][3];
	int A3[3][3];
	int A4[3][3];
	int col = 0;
	int row = 1;
	for (int i = 0; i < 3; i++)
	{

		col = 1;
		for (int j = 0; j < 3; j++)
		{
			A1[i][j] = A[row][col];
			col++;
		}
		col = 0;
		row++;
	}
	col = 0;
	row = 1;
	for (int i = 0; i < 3; i++)
	{
		for (int j = 0; j < 3; j++)
		{
			if (col == 1)
				col = 2;
			A2[i][j] = A[row][col];
			col++;
		}
		col = 0;
		row++;
	}
	col = 0;
	row = 1;
	for (int i = 0; i < 3; i++)
	{
		for (int j = 0; j < 3; j++)
		{
			if (col == 2)
				col = 3;
			A3[i][j] = A[row][col];
			col++;
		}
		col = 0;
```

```
                row++;
        }
        col = 0;
        row = 1;
        for (int i = 0; i < 3; i++)
        {
                for (int j = 0; j < 3; j++)
                {
                        A4[i][j] = A[row][col];
                        col++;
                }
                col = 0;
                row++;
        }
        int det = +(A[0][0] * deteminant4Handler(A1)) - (A[0][1] * deteminant4Handler(A2)) + (A[0][2] *
deteminant4Handler(A3)) - (A[0][3] * deteminant4Handler(A4));
        return det;
}
int** Matrix::adjugate()
{
        if (sizeOfMatrix == 2)
        {
                adjugate2x2();
        }
        else if (sizeOfMatrix == 3)
        {
                adjugate3x3();
        }
        else
        {
                adjugate4x4();
        }
        return Adjugate_Matrix;
}
void Matrix::adjugate2x2()
{
        int temp;
        Adjugate_Matrix[0][0] = A[0][0];
        Adjugate_Matrix[0][1] = A[0][1];
        Adjugate_Matrix[1][0] = A[1][0];
        Adjugate_Matrix[1][1] = A[1][1];

        temp = Adjugate_Matrix[1][1];
        Adjugate_Matrix[1][1] = Adjugate_Matrix[0][0];
        Adjugate_Matrix[0][0] = temp;
        temp = Adjugate_Matrix[1][0];
```

```
		Adjugate_Matrix[1][0] = -Adjugate_Matrix[0][1];
		Adjugate_Matrix[0][1] = -temp;

}
void Matrix::adjugate3x3()
{
		Adjugate_Matrix[0][0] = ((A[1][1] * A[2][2]) - (A[2][1] * A[1][2]));
		Adjugate_Matrix[0][1] = -((A[1][0] * A[2][2]) - (A[2][0] * A[1][2]));
		Adjugate_Matrix[0][2] = ((A[1][0] * A[2][1]) - (A[2][0] * A[1][1]));
		Adjugate_Matrix[1][0] = -((A[0][1] * A[2][2]) - (A[2][1] * A[0][2]));
		Adjugate_Matrix[1][1] = ((A[0][0] * A[2][2]) - (A[2][0] * A[0][2]));
		Adjugate_Matrix[1][2] = -((A[0][0] * A[2][1]) - (A[2][0] * A[0][1]));
		Adjugate_Matrix[2][0] = ((A[0][1] * A[1][2]) - (A[0][2] * A[1][1]));
		Adjugate_Matrix[2][1] = -((A[0][0] * A[1][2]) - (A[1][0] * A[0][2]));
		Adjugate_Matrix[2][2] = ((A[0][0] * A[1][1]) - (A[1][0] * A[0][1]));

}
void Matrix::adjugate4x4()
{
		int Cell00[3][3];
		int row = 0;
		int col = 0;
		for (int i = 0; i < 3; i++)
		{
			for (int j = 0; j < 3; j++)
			{

				if (row == 0)
					row = 1;
				if (col == 0)
					col = 1;
				Cell00[i][j] = A[row][col];
				col++;
			}
			col = 0;
			row++;
		}
		row = 0;
		col = 0;
		int Cell01[3][3];
		for (int i = 0; i < 3; i++)
		{
			for (int j = 0; j < 3; j++)
			{
				if (row == 0)
					row = 1;
```

```
                    if (col == 1)
                        col = 2;
                    Cell01[i][j] = A[row][col];
                    col++;
            }
            col = 0;
            row++;
    }
    row = 0;
    col = 0;
    int Cell02[3][3];
    for (int i = 0; i < 3; i++)
    {
            for (int j = 0; j < 3; j++)
            {
                    if (col == 2)
                        col = 3;
                    if (row == 0)
                        row = 1;
                    Cell02[i][j] = A[row][col];
                    col++;
            }
            col = 0;
            row++;
    }
    row = 0;
    col = 0;
    int Cell03[3][3];
    for (int i = 0; i < 3; i++)
    {
            for (int j = 0; j < 3; j++)
            {
                    if (row == 0)
                        row = 1;
                    Cell03[i][j] = A[row][col];
                    col++;
            }
            col = 0;
            row++;
    }
    row = 0;
    col = 0;
    int Cell10[3][3];
    for (int i = 0; i < 3; i++)
    {
            for (int j = 0; j < 3; j++)
```

```
		{
			if (col == 0)
				col = 1;
			if (row == 1)
				row = 2;
			Cell10[i][j] = A[row][col];
			col++;
		}
		col = 0;
		row++;
	}
	row = 0;
	col = 0;
	int Cell11[3][3];
	for (int i = 0; i < 3; i++)
	{
		for (int j = 0; j < 3; j++)
		{
			if (col == 1)
				col = 2;
			if (row == 1)
				row = 2;
			Cell11[i][j] = A[row][col];
			col++;
		}
		col = 0;
		row++;
	}
	row = 0;
	col = 0;
	int Cell12[3][3];
	for (int i = 0; i < 3; i++)
	{
		for (int j = 0; j < 3; j++)
		{
			if (col == 2)
				col = 3;
			if (row == 1)
				row = 2;
			Cell12[i][j] = A[row][col];
			col++;
		}
		col = 0;
		row++;
	}
	row = 0;
```

```
col = 0;
int Cell13[3][3];
for (int i = 0; i < 3; i++)
{
        for (int j = 0; j < 3; j++)
        {
                if (row == 1)
                        row = 2;
                Cell13[i][j] = A[row][col];
                col++;
        }
        col = 0;
        row++;
}
int Cell20[3][3];
row = 0;
col = 0;
for (int i = 0; i < 3; i++)
{
        for (int j = 0; j < 3; j++)
        {
                if (row == 2)
                        row = 3;
                if (col == 0)
                        col = 1;
                Cell20[i][j] = A[row][col];
                col++;
        }
        col = 0;
        row++;
}
int Cell21[3][3];
row = 0;
col = 0;
for (int i = 0; i < 3; i++)
{
        for (int j = 0; j < 3; j++)
        {
                if (row == 2)
                        row = 3;
                if (col == 1)
                        col = 2;
                Cell21[i][j] = A[row][col];
                col++;
        }
        col = 0;
```

```
            row++;
}
int Cell22[3][3];
row = 0;
col = 0;
for (int i = 0; i < 3; i++)
{
            for (int j = 0; j < 3; j++)
            {

                        if (row == 2)
                                    row = 3;
                        if (col == 2)
                                    col = 3;
                        Cell22[i][j] = A[row][col];
                        col++;
            }
            col = 0;
            row++;
}
int Cell23[3][3];
row = 0;
col = 0;
for (int i = 0; i < 3; i++)
{
            for (int j = 0; j < 3; j++)
            {
                        if (row == 2)
                                    row = 3;

                        Cell23[i][j] = A[row][col];
                        col++;
            }
            col = 0;
            row++;
}
int Cell30[3][3];
row = 0;
col = 0;
for (int i = 0; i < 3; i++)
{
            for (int j = 0; j < 3; j++)
            {
                        if (col == 0)
                                    col = 1;
                        Cell30[i][j] = A[row][col];
```

```
                col++;
        }
        col = 0;
        row++;
}
int Cell31[3][3];
row = 0;
col = 0;
for (int i = 0; i < 3; i++)
{
        for (int j = 0; j < 3; j++)
        {
                if (col == 1)
                        col = 2;
                Cell31[i][j] = A[row][col];
                col++;
        }
        col = 0;
        row++;
}
int Cell32[3][3];
row = 0;
col = 0;
for (int i = 0; i < 3; i++)
{
        for (int j = 0; j < 3; j++)
        {
                if (col == 2)
                        col = 3;

                Cell32[i][j] = A[row][col];
                col++;
        }
        col = 0;
        row++;
}
int Cell33[3][3];
row = 0;
col = 0;
for (int i = 0; i < 3; i++)
{
        for (int j = 0; j < 3; j++)
        {
                Cell33[i][j] = A[row][col];
                col++;
        }
```

```
                col = 0;
                row++;
        }
        Adjugate_Matrix[0][0] = +Matrix::deteminant4Handler(Cell00);
        Adjugate_Matrix[0][1] = -Matrix::deteminant4Handler(Cell01);
        Adjugate_Matrix[0][2] = +Matrix::deteminant4Handler(Cell02);
        Adjugate_Matrix[0][3] = -Matrix::deteminant4Handler(Cell03);
        Adjugate_Matrix[1][0] = -Matrix::deteminant4Handler(Cell10);
        Adjugate_Matrix[1][1] = +Matrix::deteminant4Handler(Cell11);
        Adjugate_Matrix[1][2] = -Matrix::deteminant4Handler(Cell12);
        Adjugate_Matrix[1][3] = +Matrix::deteminant4Handler(Cell13);
        Adjugate_Matrix[2][0] = +Matrix::deteminant4Handler(Cell20);
        Adjugate_Matrix[2][1] = -Matrix::deteminant4Handler(Cell21);
        Adjugate_Matrix[2][2] = +Matrix::deteminant4Handler(Cell22);
        Adjugate_Matrix[2][3] = -Matrix::deteminant4Handler(Cell23);
        Adjugate_Matrix[3][0] = -Matrix::deteminant4Handler(Cell30);
        Adjugate_Matrix[3][1] = +Matrix::deteminant4Handler(Cell31);
        Adjugate_Matrix[3][2] = -Matrix::deteminant4Handler(Cell32);
        Adjugate_Matrix[3][3] = +Matrix::deteminant4Handler(Cell33);
}
int** Matrix::transpose()
{
        if (sizeOfMatrix == 2)
        {
                transpose2x2();
        }
        else if (sizeOfMatrix == 3)
        {
                transpose3x3();
        }
        else
        {
                transpose4x4();
        }
        return A_transpose;
}
void Matrix::transpose2x2()
{
        A_transpose[0][0] = Adjugate_Matrix[0][0];
        A_transpose[0][1] = Adjugate_Matrix[0][1];
        A_transpose[1][0] = Adjugate_Matrix[1][0];
        A_transpose[1][1] = Adjugate_Matrix[1][1];
        int temp = A_transpose[0][1];
        A_transpose[0][1] = A_transpose[1][0];
        A_transpose[1][0] = temp;
}
```

```
void Matrix::transpose3x3()
{
        A_transpose[0][0] = Adjugate_Matrix[0][0];
        A_transpose[0][1] = Adjugate_Matrix[0][1];
        A_transpose[0][2] = Adjugate_Matrix[0][2];
        A_transpose[1][0] = Adjugate_Matrix[1][0];
        A_transpose[1][1] = Adjugate_Matrix[1][1];
        A_transpose[1][2] = Adjugate_Matrix[1][2];
        A_transpose[2][0] = Adjugate_Matrix[2][0];
        A_transpose[2][1] = Adjugate_Matrix[2][1];
        A_transpose[2][2] = Adjugate_Matrix[2][2];
        int temp1;
        int temp2;
        temp1 = A_transpose[0][1];
        temp2 = A_transpose[0][2];
        A_transpose[0][1] = A_transpose[1][0];
        A_transpose[0][2] = A_transpose[2][0];
        A_transpose[1][0] = temp1;
        A_transpose[2][0] = temp2;
        temp2 = A_transpose[1][2];
        A_transpose[1][2] = A_transpose[2][1];
        A_transpose[2][1] = temp2;
}
void Matrix::transpose4x4()
{
        A_transpose[0][0] = Adjugate_Matrix[0][0];
        A_transpose[0][1] = Adjugate_Matrix[0][1];
        A_transpose[0][2] = Adjugate_Matrix[0][2];
        A_transpose[0][3] = Adjugate_Matrix[0][3];
        A_transpose[1][0] = Adjugate_Matrix[1][0];
        A_transpose[1][1] = Adjugate_Matrix[1][1];
        A_transpose[1][2] = Adjugate_Matrix[1][2];
        A_transpose[1][3] = Adjugate_Matrix[1][3];
        A_transpose[2][0] = Adjugate_Matrix[2][0];
        A_transpose[2][1] = Adjugate_Matrix[2][1];
        A_transpose[2][2] = Adjugate_Matrix[2][2];
        A_transpose[2][3] = Adjugate_Matrix[2][3];
        A_transpose[3][0] = Adjugate_Matrix[3][0];
        A_transpose[3][1] = Adjugate_Matrix[3][1];
        A_transpose[3][2] = Adjugate_Matrix[3][2];
        A_transpose[3][3] = Adjugate_Matrix[3][3];
        int temp1;
        int temp2;
        int temp3;
        temp1 = A_transpose[1][0];
        temp2 = A_transpose[2][0];
```

```
		temp3 = A_transpose[3][0];
		A_transpose[1][0] = A_transpose[0][1];
		A_transpose[2][0] = A_transpose[0][2];
		A_transpose[3][0] = A_transpose[0][3];
		A_transpose[0][1] = temp1;
		A_transpose[0][2] = temp2;
		A_transpose[0][3] = temp3;
		temp1 = A_transpose[3][1];
		temp2 = A_transpose[3][2];
		A_transpose[3][1] = A_transpose[1][3];
		A_transpose[3][2] = A_transpose[2][3];

		A_transpose[1][3] = temp1;
		A_transpose[2][3] = temp2;
}
double** Matrix::inverseMatrix()
{
		int deteminant = Matrix::deteminant();
		for (int i = 0; i < sizeOfMatrix; i++)
		{
			A_inverse[i] = new double[sizeOfMatrix];
		}
		for (int i = 0; i < sizeOfMatrix; i++)
		{
			for (int j = 0; j < sizeOfMatrix; j++)
			{
				A_inverse[i][j] = (1.0 / deteminant) * A_transpose[i][j];
			}
		}
		return A_inverse;
}
void Matrix::displayMatrix()
{
		std::cout << "A Matrix data" << std::endl;
		std::cout << "--------------------------------------" << std::endl;
		for (int i = 0; i < sizeOfMatrix; i++)
		{
			for (int j = 0; j < sizeOfMatrix; j++)
			{
				if (j == 0)
				{
					std::cout << "|";
				}
				std::cout << " " << A[i][j];
				if (j == (sizeOfMatrix - 1))
				{
```

```
                                        std::cout << " |";
                                }
                        }
                        std::cout << std::endl;
                }
}
void Matrix::displayAdjugate()
{
        std::cout << "Adjugate Matrix of A" << std::endl;
        std::cout << "--------------------------------------" << std::endl;
        for (int i = 0; i < sizeOfMatrix; i++)
        {
                for (int j = 0; j < sizeOfMatrix; j++)
                {
                        if (j == 0)
                        {
                                std::cout << "|";
                        }
                        std::cout << " " << Adjugate_Matrix[i][j];
                        if (j == (sizeOfMatrix - 1))
                        {
                                std::cout << " |";
                        }
                }
                std::cout << std::endl;
        }
}
void Matrix::displayInverse()
{
        std::cout << "Inverse Matrix of A" << std::endl;
        std::cout << "--------------------------------------" << std::endl;
        for (int i = 0; i < sizeOfMatrix; i++)
        {
                for (int j = 0; j < sizeOfMatrix; j++)
                {
                        if (j == 0)
                        {
                                std::cout << "|";
                        }
                        std::cout << " " << A_inverse[i][j];
                        if (j == (sizeOfMatrix - 1))
                        {
                                std::cout << " |";
                        }
                }
                std::cout << std::endl;
```

```
		}
}
void Matrix::displayTranspose()
{
		std::cout << "Tranpose of cofactor Matrix" << std::endl;
		std::cout << "-------------------------------------" << std::endl;
		for (int i = 0; i < sizeOfMatrix; i++)
		{
			for (int j = 0; j < sizeOfMatrix; j++)
			{
				if (j == 0)
				{
					std::cout << "|";
				}
				std::cout << " " << A_transpose[i][j];
				if (j == (sizeOfMatrix - 1))
				{
					std::cout << " |";
				}
			}
			std::cout << std::endl;
		}
}
void Matrix::displayAnswer()
{
		std::cout << "Computation result" << std::endl;
		std::cout << "-------------------------------------" << std::endl;
		for (int i = 0; i < sizeOfMatrix; i++)
		{
			std::cout << "| " << result[i] << " |" << std::endl;
		}
}
void Matrix::displayBmatrix()
{
		std::cout << "B matrix data" << std::endl;
		std::cout << "-------------------------------------" << std::endl;
		for (int i = 0; i < sizeOfMatrix; i++)
		{
			std::cout << "| " << B[i] << " |" << std::endl;
		}
}
void Matrix::ComputeAnswer()
{
		for (int i = 0; i < sizeOfMatrix; i++)
		{
			result[i] = 0;
```

```
		}
		for (int i = 0; i < sizeOfMatrix; i++)
		{
			for (int j = 0; j < sizeOfMatrix; j++)
			{
				result[i] += (A_inverse[i][j] * B[j]);
			}
		}
}
```

Main program code (main.cpp file)

```
#include<iostream>
#include<iomanip>
#include<vector>
#include"Matrix.h"
using namespace std;
int main()
{
	while (true)//program loop
	{
		cout << "Enter matrix size: ";
		int mat_size;
		cin >> mat_size;
		cout << endl;
		cout << "Enter values for " << mat_size << "x" << mat_size << " matrix" << endl;
		cout << "---------------------------------------" << endl;
		int** mat;
		mat = new int*[mat_size];
		for (int i = 0; i < mat_size; i++)
		{
			mat[i] = new int[mat_size];
		}
		int* col_matrix = new int[mat_size];
		for (int i = 0; i < mat_size; i++)
		{
			for (int j = 0; j < mat_size; j++)
			{
				cout << "Enter value for cell " << i << j << ": ";
				cin >> mat[i][j];
				cout << endl;
			}
		}

		cout << "Enter column matrix data" << endl;
		cout << "---------------------------------------" << endl;
		for (int i = 0; i < mat_size; i++)
```

```
		{
			cout << "Enter value for cell " << i << ": ";
			cin >> col_matrix[i];
			cout << endl;
		}

		Matrix *matrix = new Matrix(mat_size, mat, col_matrix);
		matrix->displayMatrix();
		matrix->displayBmatrix();
		cout << "Matrix determinant = " << matrix->deteminant() << endl;
		matrix->displayAdjugate();
		matrix->displayTranspose();
		matrix->displayInverse();
		matrix->displayAnswer();
		//free up memory
		matrix->~Matrix();
		cout << "-------------------------------------" << endl;
		cout << "Enter any key to calculate another or 1 to exit " << endl;
		cout << "-------------------------------------" << endl;
		cout << ">>";
		char c;
		cin >> c;
		if (c == '1')
			break;
		else
		{
			cout << "-------------------------------------" << endl;
		}
	}

}
```

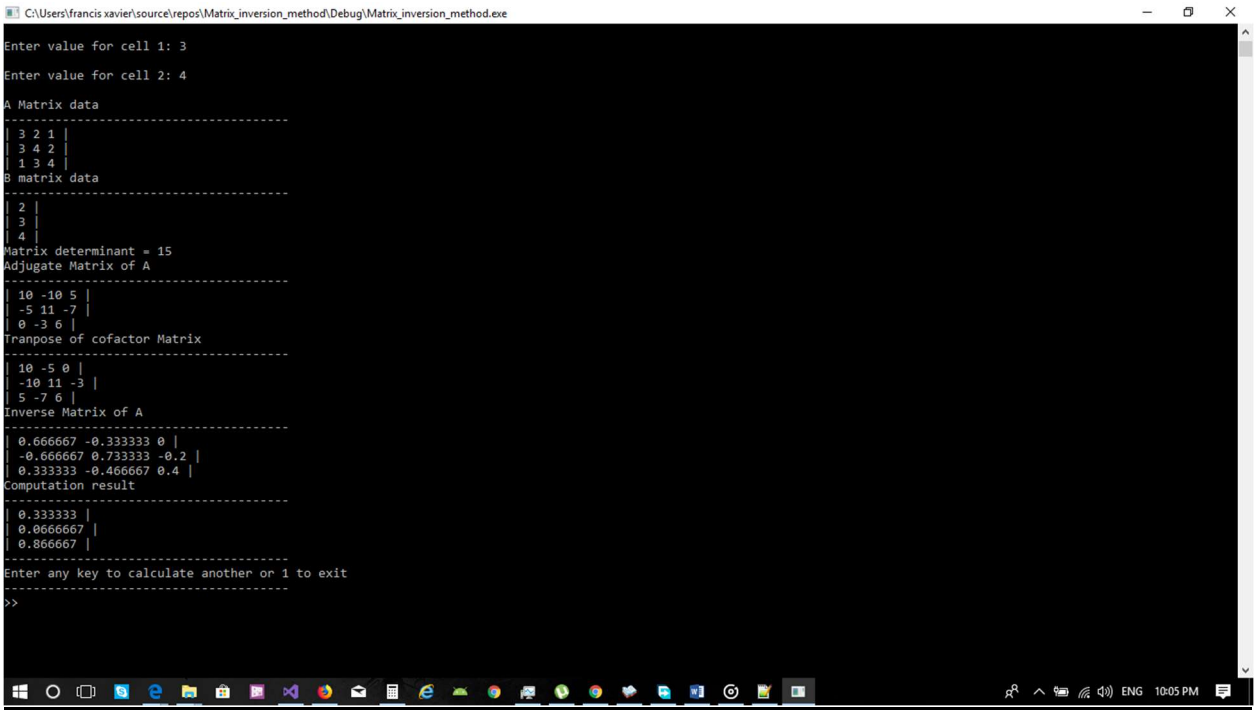

FIG 2: Output Screen Shots

Using Python Programming Language to solve a system of linear equations by Matrix Inversion Method.

Consider
a11x1 + a12x2 = b1
a21x1 + a22x2 = b2

$$\begin{vmatrix} a11 & a12 \\ a21 & a22 \end{vmatrix} \quad \begin{vmatrix} x1 \\ x2 \end{vmatrix} \quad \begin{vmatrix} b1 \\ b2 \end{vmatrix}$$

A X B

Which is of the form AX = B, if A-1 exists then the solution is X = A-1B. Using Python and NumPy this can be translated into a program;

Python is an interpreted high-level programming language for general-purpose programming.
NumPy is the fundamental package for scientific computing in python. It is a Python library that provides:

1. An array object of arbitrary homogeneous items.
2. Fast mathematical operations over arrays.
3. Linear Algebra, Fourier Transforms, Random Number Generation.

Source Code:

```
import numpy as np


#Here Matrix A is entered into the Program...

print("\nEnter the values for Matrix A by entering each row one by one, \nthen press enter after a row is completed. \nTo terminate press Enter without any value inputed to a line...")

a = []#Initializes a list called 'a'

while True:

    r = input("\nNext Row >> ")

    if not r.strip(): #empty input

        break

    a.append(list(map(int, r.split())))#Adds integer element to the end of List 'a'

print("\nMatrix A = ", a)
```

- NumPy is imported into the source code to allow for mathematical operations over arrays.
- The above code snippet is used to prompt users for input, this allows the user enter any type of matrix (2X2, 3X3, 4X4) into the program by entering the values in each row on a line before pressing the enter key. These values are then stored onto the array a.

- a.append(list(map(int, r.split()))) adds element to the end of array a, this statement is what adds users input into array a.

```
#defines Matrix A

A = np.array(a)


#defines Matrix B

B = np.array(b)


#linalg.inv is a NumPy function to find the inverse of matrix A

C = np.linalg.inv(A)

print("\nInverse of A is: \n", C)


#np.dot is a Numpy function to find dot product of Inverse matrix A and B

D = np.dot(C, B)

print("\nDot Product is: \n", D)
```

- A = np.array(a) and B = np.array(b) are used to define the arrays a and b into matrices A and B whereby computations can be performed on them.
- C = np.linalg.inv(A) is used to compute the inverse of A (A^{-1}) and store the value in C
- D = np.dot(C, B) is used to compute the dot product of matrix C (inverse of A) and matrix B and stores the values in D.

```
a = len(D)#Returns the number of elements in the list.

i = 1

b = a - a

while i != a + 1:

    word = "X" + str(i)

    print("\n", word, "is", D[b])#Prints out the result one at a time.

    i = i + 1

    b = b + 1
```

- len(D) returns the number of elements in the list/array and this is used to determine the amount of outputs (X1, X2...) that will be displayed using a while loop.

4. PYTHON CODE FOR MATRIX INVERSION METHOD

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Jan 30 19:52:29 2018
@author: Daramola Oluwafunto
"""
import numpy as np
#Defines a function to divide various parts of program execution.
def draw_line(n, ch):
    for line in range(n):
        print(ch, end="")
#Here Matrix A is entered into the Program...
print("\nEnter the values for Matrix A by entering each row one by one, \nthen press enter after a row is
completed. \nTo terminate press Enter without any value inputed to a line...")
draw_line(80,'-')
a = []#Initializes a list called 'a'
while True:
    r = input("\nNext Row >> ")
    draw_line(80,'~') #Calls the function draw_line
    if not r.strip(): #empty input
        break
    a.append(list(map(int, r.split())))#Adds integer element to the end of List 'a'
print("\nMatrix A = ", a)
draw_line(80,'-')#Calls the function draw_line
#Here Matrix B is entered into the Program
print("\nEnter the values for Matrix B by entering each row one by one, \nthen press enter after a row is
completed. \nTo terminate press Enter without any value inputed to a line...")
draw_line(80,'-')
b = []#Initializes a list called 'a'
while True:
    r = input("\nNext Row >> ")
    draw_line(80,'~')#Calls the function draw_line
    if not r.strip(): #empty input
        break
    b.append(list(map(int, r.split())))#Adds integer element to the end of List 'b'
print("\nMatrix B = ", b)
draw_line(80,'-')#Calls the function draw_line
#defines Matrix A
A = np.array(a)
#defines Matrix B
B = np.array(b)
#linalg.inv is a NumPy function to find the inverse of matrix A
C = np.linalg.inv(A)
print("\nInverse of A is: \n", C)
```

```
#np.dot is a Numpy function to find dot product of Inverse matrix A and B
D = np.dot(C, B)
print("\nDot Product is: \n", D)
draw_line(80,'-')#Calls the function draw_line
a = len(D)#Returns the number of elements in the list.
i = 1
b = a - a
while i != a + 1:
    word = "X" + str(i)
    print("\n", word, "is", D[b])#Prints out the result one at a time.
    i = i + 1
    b = b + 1
draw_line(80,'-')#Calls the function draw_line
r = input("\nPress any Key to Quit...")
```

5. CONCLUSION

Matrix inversion method is a simple method for solving a series of simultaneous equations but when the variables exceed 3 the solution become very long and stressful to compute, it is a very good method for calculating variables less than 4. However, for simultaneous equations with 4 or more variables, we recommend a different numerical method like Gauss-Jordan elimination method.

REFERENCES

1. Wikipedia (Adjugate matrix): https://en.wikipedia.org/wiki/Adjugate_matrix
2. Wikipedia(Minor): https://en.wikipedia.org/wiki/Minor_(linear_algebra)
3. Wikipedia(Transpose): https://en.wikipedia.org/wiki/Minor_(linear_algebra)
4. Math Captain(Matrix transpose): http://www.mathcaptain.com/algebra/matrix-transpose.html
5. Math world(inverse matrix): http://www.mathwords.com/i/inverse_of_a_matrix.htm
6. Techopedia (C++ Programming Language): https://www.techopedia.com/definition/26184/c-programming-language