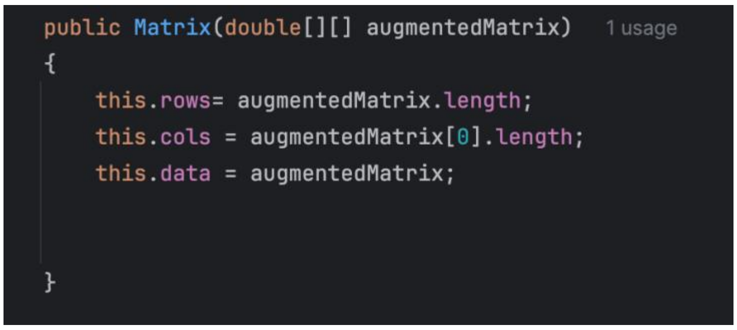
Criteria C

Table of Contents

Word Count: 1000

1.IDE used:



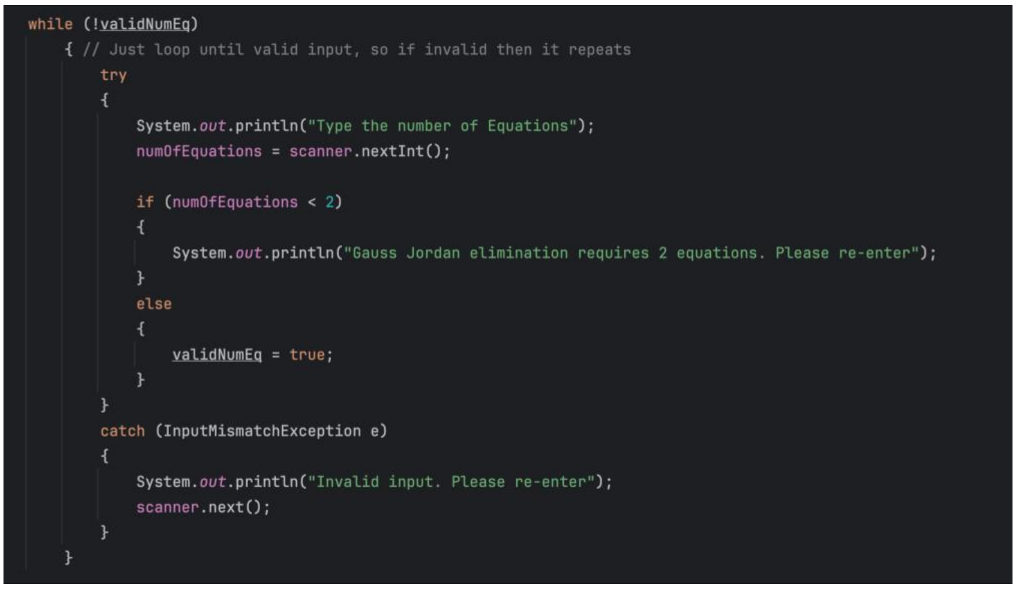
```java
public Matrix(double[][] augmentedMatrix)    1 usage
{
    this.rows= augmentedMatrix.length;
    this.cols = augmentedMatrix[0].length;
    this.data = augmentedMatrix;


}
```

Figure 1:JDK feature in IntelliJ IDEA

IntelliJ IDEA was used to create this product. This is because the client was used to using it and it has some features in its Java development kit (JDK). Such examples would be that it tells the number of usage for methods allowing me to remove unnecessary methods.

2.InputHandler:

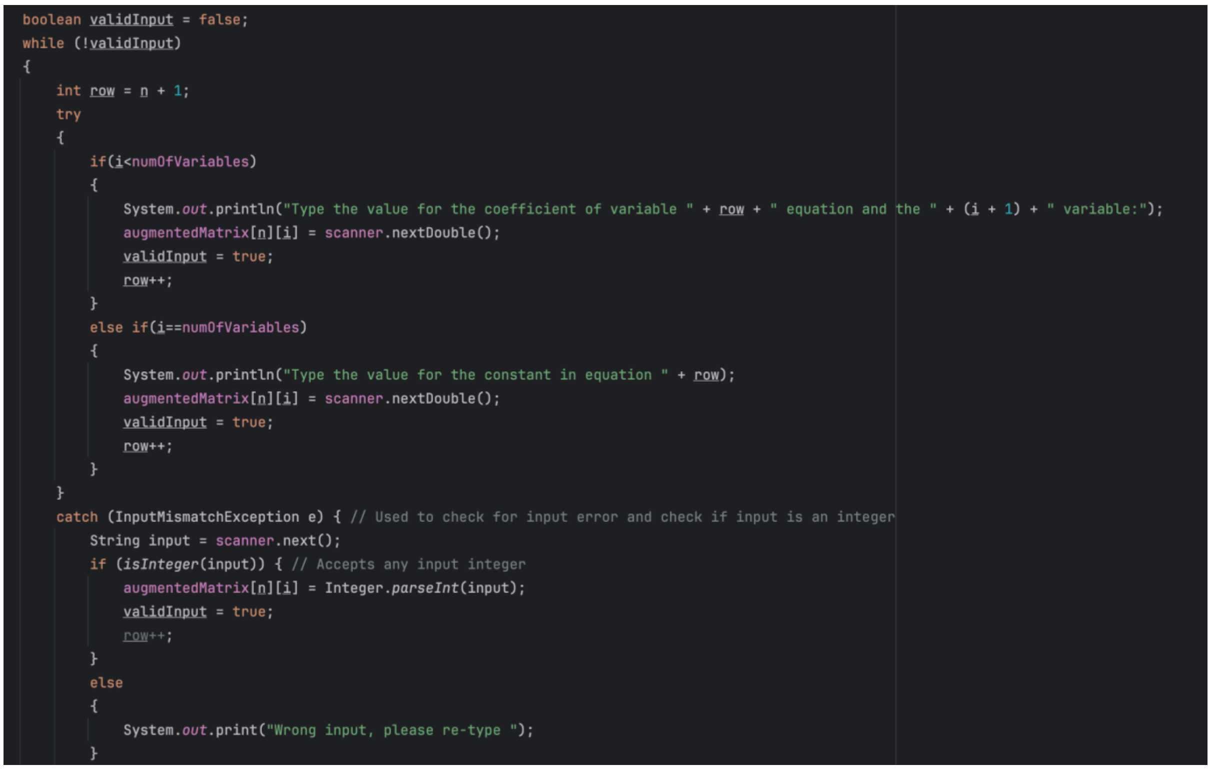
a.Scanner and user input validation:



```java
while (!validNumEq)
    { // Just loop until valid input, so if invalid then it repeats
        try
        {
            System.out.println("Type the number of Equations");
            numOfEquations = scanner.nextInt();

            if (numOfEquations < 2)
            {
                System.out.println("Gauss Jordan elimination requires 2 equations. Please re-enter");
            }
            else
            {
                validNumEq = true;
            }
        }
        catch (InputMismatchException e)
        {
            System.out.println("Invalid input. Please re-enter");
            scanner.next();
        }
    }
```

Figure 2: Case for specifying the number of equations

A try and catch is implemented when inputting the number of equations to construct the 2d array augmented matrix. A while loop is implemented with the condition of validNumEq being false, this allows the program to keep asking the user for a valid number of equations until the user does. Once he does ValidNumEq becomes true and the program proceeds to ask for the number of variables, where a similar structure is created.



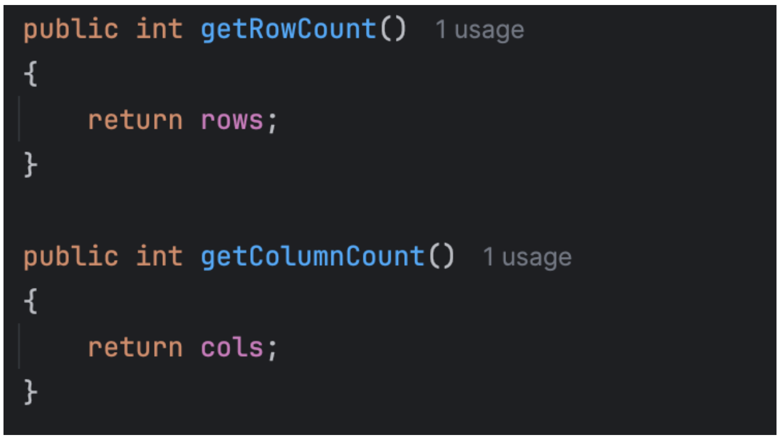
```java
boolean validInput = false;
while (!validInput)
{
    int row = n + 1;
    try
    {
        if(i<numOfVariables)
        {
            System.out.println("Type the value for the coefficient of variable " + row + " equation and the " + (i + 1) + " variable:");
            augmentedMatrix[n][i] = scanner.nextDouble();
            validInput = true;
            row++;
        }
        else if(i==numOfVariables)
        {
            System.out.println("Type the value for the constant in equation " + row);
            augmentedMatrix[n][i] = scanner.nextDouble();
            validInput = true;
            row++;
        }
    }
    catch (InputMismatchException e) { // Used to check for input error and check if input is an integer
        String input = scanner.next();
        if (isInteger(input)) { // Accepts any input integer
            augmentedMatrix[n][i] = Integer.parseInt(input);
            validInput = true;
            row++;
        }
        else
        {
            System.out.print("Wrong input, please re-type ");
        }
```

Figure 3: Handling inputs for elements for augmented matrix

The Scanner object is an attribute of the inputHandler class and is instatntaied for each value input in the augmentedMatrix. A nested for loop not shown in figure 2 goes through each element in the augmentedMatrix. When a user inputs an invalid data type such as Strings, the program will catch the exception and check whether the input is a integer. If it is not then it will display a message to re-input a correct data type which is integer or double. If the input is an integer then it will be parsed into a double. Once all of the elements is created the augmented matrix is returned to the method which is called MatrixConstructor.

3.Matrix manipulation methods

a.Accessors:



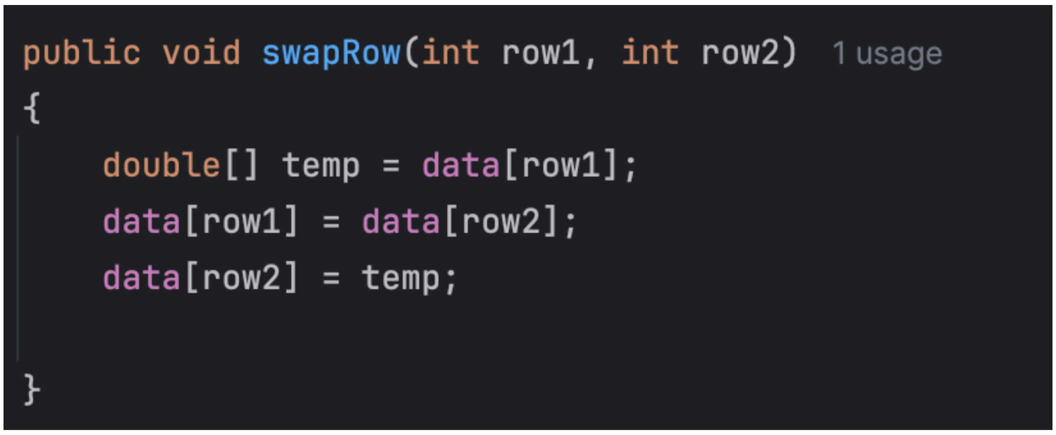
```
public int getRowCount()  1 usage
{
    return rows;
}

public int getColumnCount()  1 usage
{
    return cols;
}
```

Figure 4: Accessors in Matrix class

In our project, we encapsulate matrix data and operations in the Matrix class. The getRowCount() and getColumnCount() serve as accessors. They abstract internal representation of matrix and provides a simple implementation to retrieve the matrix dimensions. They are essential in the Gaussian Class which is where the RREF is calculated.

b.Mutators:

Mutators used in the RREF() method located in the Gaussian Class. They are a form of abstraction as they hide complexity in the code.

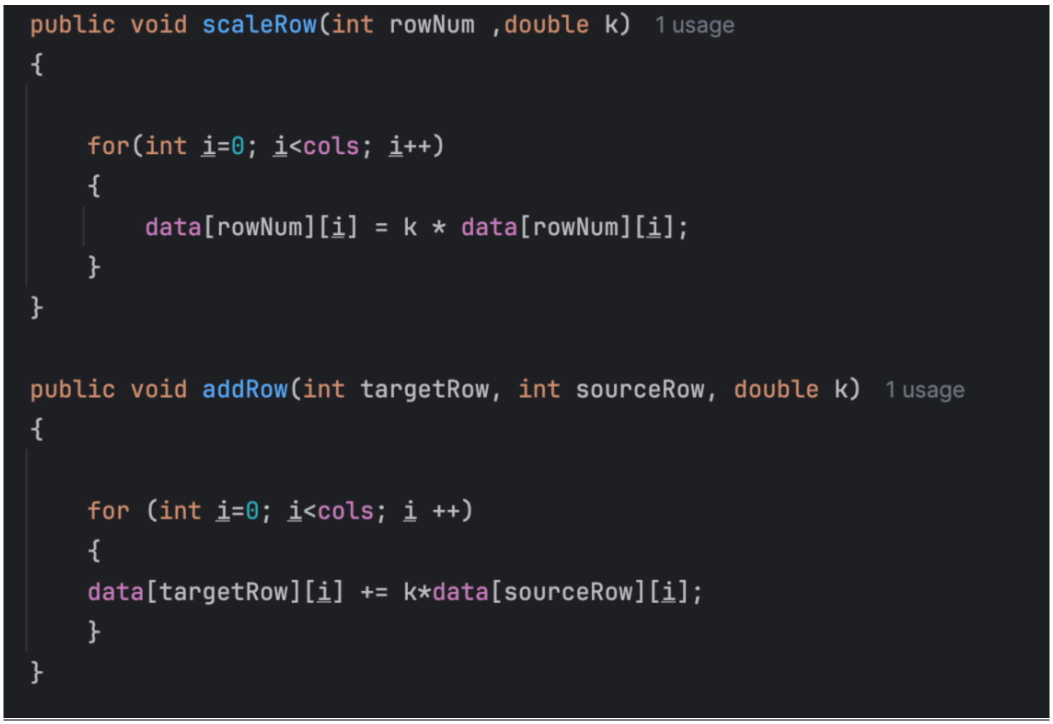


```
public void swapRow(int row1, int row2)  1 usage
{
    double[] temp = data[row1];
    data[row1] = data[row2];
    data[row2] = temp;

}
```

Figure 5: Mutators in Matrix class

The swapRow method relies on a variable 'temp' to swap the two rows. The data entries must be integers that specifiy the rows of the matrix[or data[] which is part of the Matrix class attribute.

I.For Loops:



```
public void scaleRow(int rowNum ,double k)  1 usage
{

    for(int i=0; i<cols; i++)
    {
        data[rowNum][i] = k * data[rowNum][i];
    }
}

public void addRow(int targetRow, int sourceRow, double k)  1 usage
{

    for (int i=0; i<cols; i ++)
    {
    data[targetRow][i] += k*data[sourceRow][i];
    }
}
```

Figure 6: Use of for loops

In the scaleRow and addRow methods, a for loop is implemented to traverse through the specified row of the 2d array data. In the scaleRow, the k value is multiplied to each element in the data rowNum row. The for loop in the addRow ensures that all elements in the target row are added with a constant K multipled from the initial row named ‘sourceRow’.

4.Gaussian Elimination

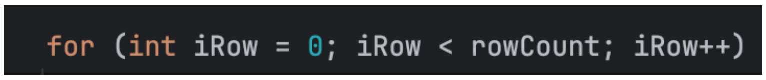
The backbone of the program is handled by the Gaussian class. It contains 4 methods where the RREF method holds the most importance of the program.

RREF:

I used the Linear Algebra by otto Bretscher textbook to learn how to find the RREF.

Nested for loop and while loop

I.Outer for loop (Row Iteration Loop):

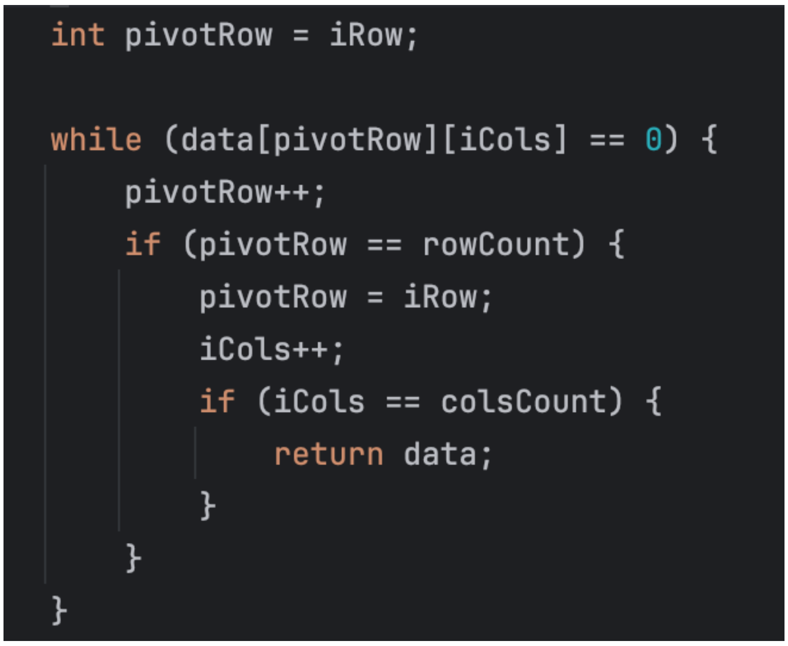


```
for (int iRow = 0; iRow < rowCount; iRow++)
```

Figure 7: Use of outer for loop

Makes sure the RREF method iterates through every row

II.Inner While loop (pivot selection):

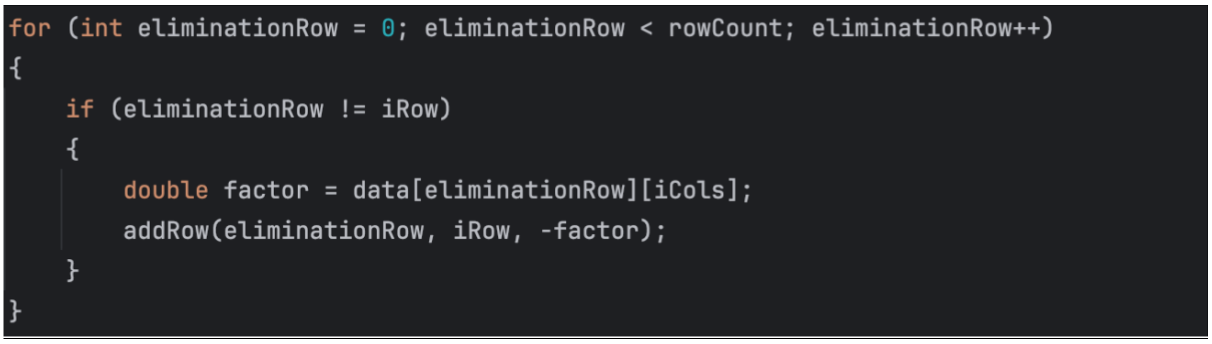


```
int pivotRow = iRow;

while (data[pivotRow][iCols] == 0) {
    pivotRow++;
    if (pivotRow == rowCount) {
        pivotRow = iRow;
        iCols++;
        if (iCols == colsCount) {
            return data;
        }
    }
}
```

Figure 8: Use of inner while loop complementing the outer for loop

This while loop searches for a nonzero pivot. The pivot is just the initial diagonal entry in data[][]. It utilizes if statements such as if the pivotRow is equal to the number of rows then pivot row goes to the current row created by the outer for loop and the currently analysed column is iterated.

III.Inner For loop (Row Elimination):

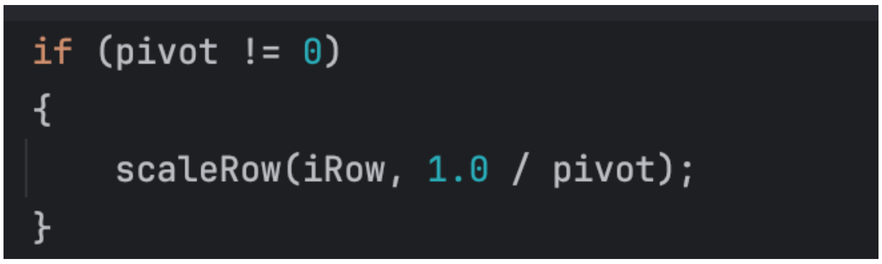


```
for (int eliminationRow = 0; eliminationRow < rowCount; eliminationRow++)
{
    if (eliminationRow != iRow)
    {
        double factor = data[eliminationRow][iCols];
        addRow(eliminationRow, iRow, -factor);
    }
}
```

Figure 9: Use of inner for loop complementing the outer for loop

After the pivot row is selected and scaled, it eliminates all of the entries in the current column that are not non-zero. To eliminate the elements in the columns that is not the pivot, it must do the operation for the entire row. I have used the addRow mutator method located in the Matrix class.

IV.If statements



```
if (pivot != 0)
{
    scaleRow(iRow, 1.0 / pivot);
}
```

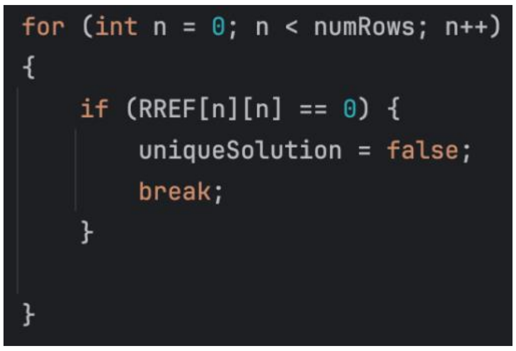
Figure 9: Use of if loop within a for loop

As per the client’s review, I cannot divide the row by 0 as it is mathematically impossible. He advised that I check if the pivot is nonzero and to scale the pivot down to 1 and do the same for the entire row. I implemented this if statements coupled with the scaleRow method located in the Matrix class.

5.ReadSolution:

As shown in pseudocode in criterion B, the readSolution() method requires traversal patterns. In this explanation I will only go over the traversal patterns as I have already explained the logic of readSolution in criterion B.

a. Diagonal Traversal:

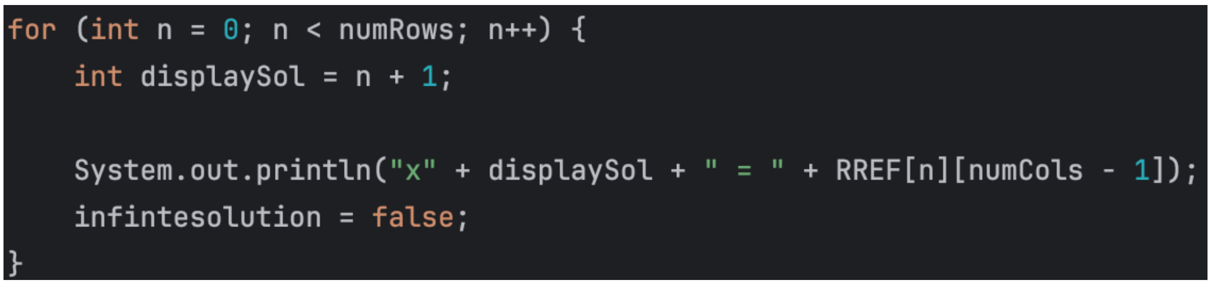


```
for (int n = 0; n < numRows; n++)
{
    if (RREF[n][n] == 0) {
        uniqueSolution = false;
        break;
    }

}
```

Figure 10: Diagonal 2d array traversal pattern

Iterates over rows to check for zero diagonal entries in pivot. The traversal only goes through the pivots.

b.Row-wise Traversal (unique solution output)

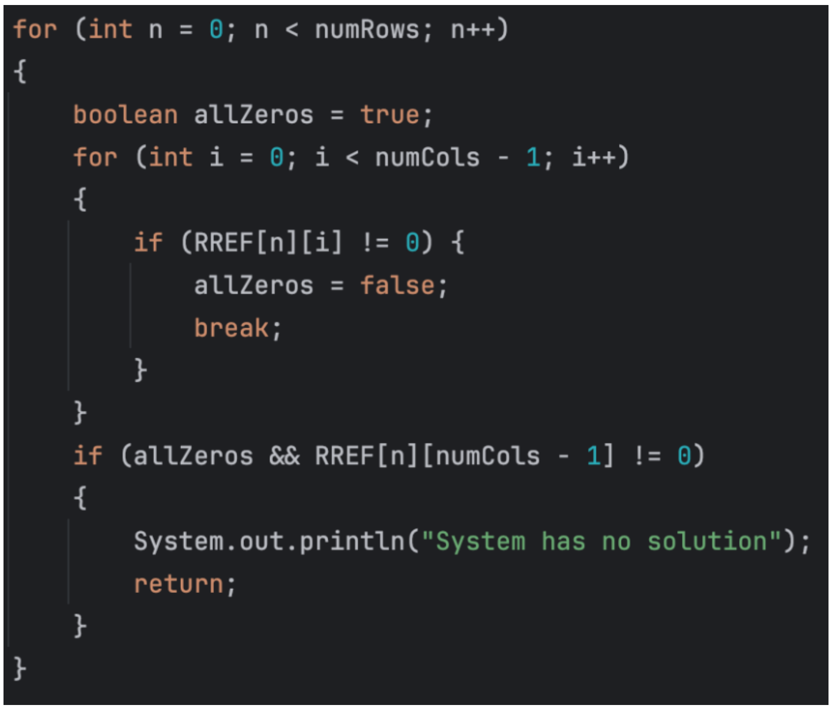


```
for (int n = 0; n < numRows; n++) {
    int displaySol = n + 1;

    System.out.println("x" + displaySol + " = " + RREF[n][numCols - 1]);
    infintesolution = false;
}
```

Figure 11: Row-wise traversal (GeeksforGeeks, n.d.)

Outputs all entries from the last column as equal to the unique solution.

c. Row and Inner Column Traversal (Inconsistency Check)

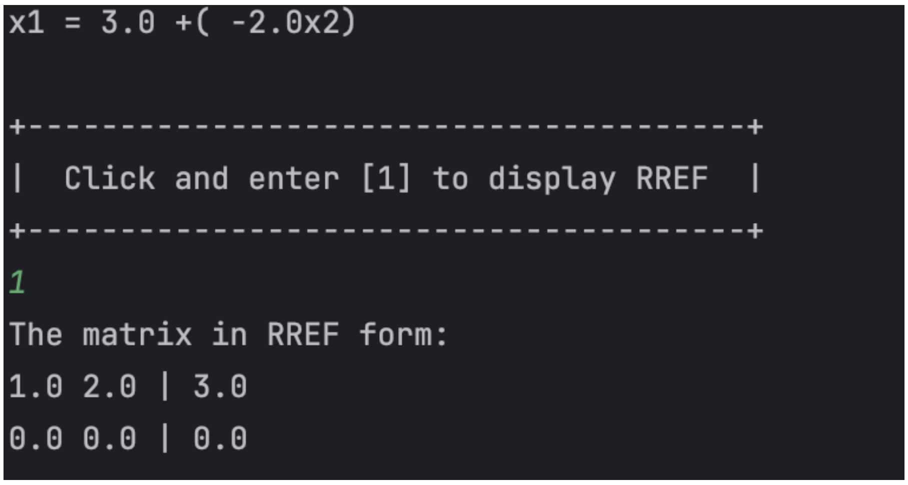


```java
for (int n = 0; n < numRows; n++)
{
    boolean allZeros = true;
    for (int i = 0; i < numCols - 1; i++)
    {
        if (RREF[n][i] != 0) {
            allZeros = false;
            break;
        }
    }
    if (allZeros && RREF[n][numCols - 1] != 0)
    {
        System.out.println("System has no solution");
        return;
    }
}
```

Figure 12: Row and inner Column Traversal (GeeksforGeeks, n.d.)

For each row, iterates through the columns to verify if there is any cases where the coefficient is a zero and the constant is not zero. Similar structure is implemented to collect free variables and display them.

6.Text-based user interface



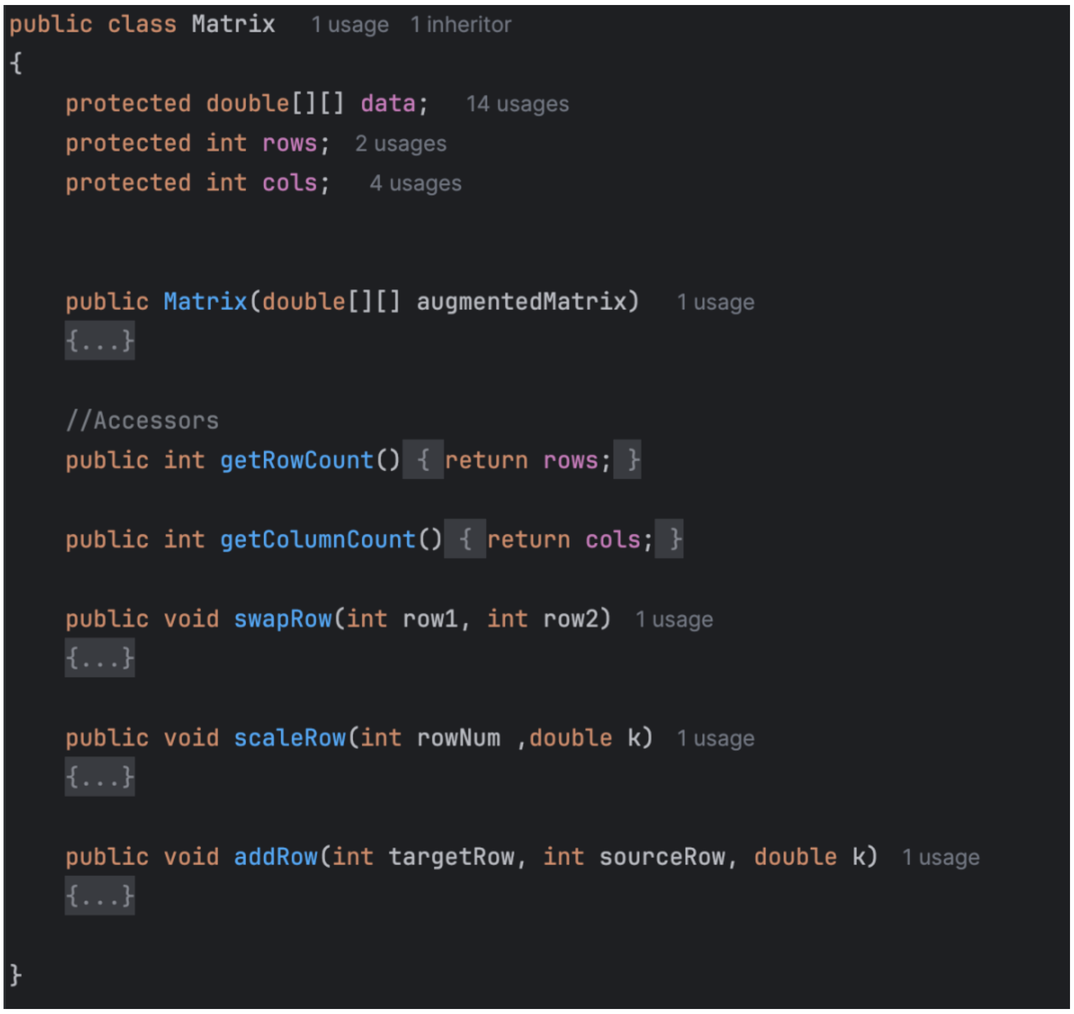
```
x1 = 3.0 +( -2.0x2)

+--------------------------------------+
|  Click and enter [1] to display RREF  |
+--------------------------------------+
1
The matrix in RREF form:
1.0 2.0 | 3.0
0.0 0.0 | 0.0
```

Figure 13: Output format

This is an example of the text-based user interface, showcasing the parametric form to the solution of the equation and the RREF form.

7.Object-Oriented Programming

a. Encapsulation:

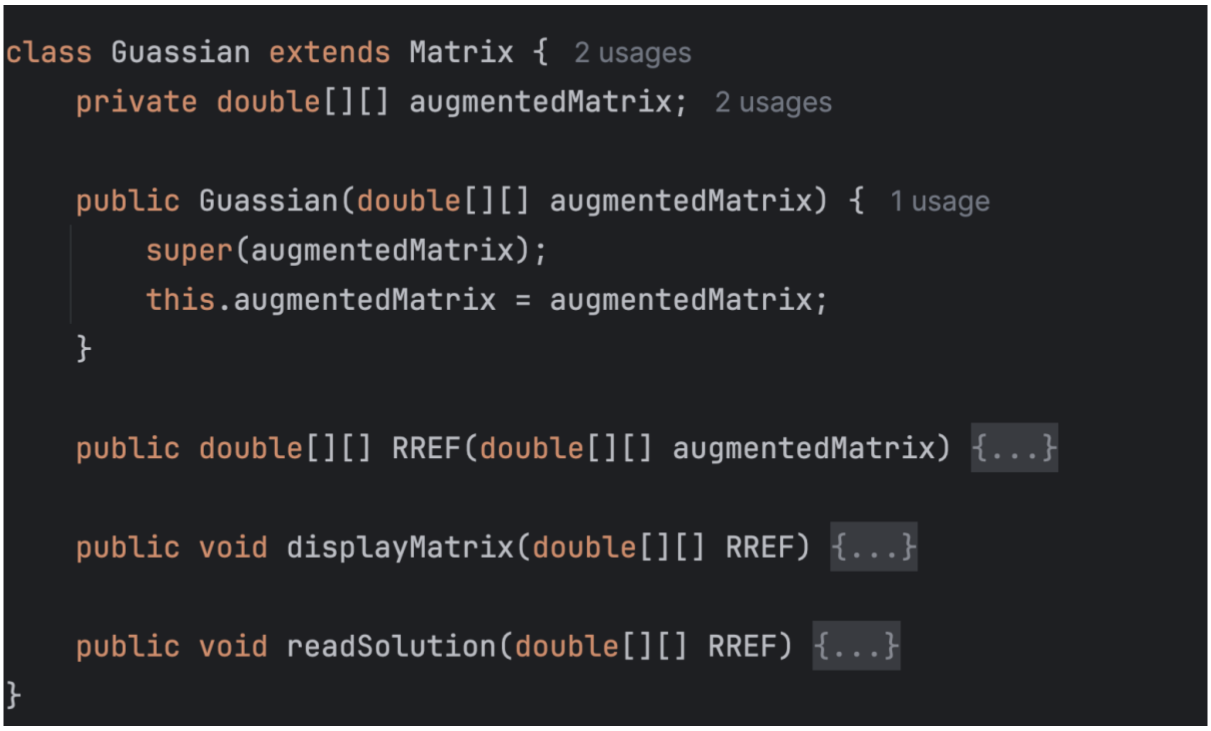


```
public class Matrix   1 usage   1 inheritor
{
    protected double[][] data;   14 usages
    protected int rows;  2 usages
    protected int cols;   4 usages


    public Matrix(double[][] augmentedMatrix)   1 usage
    {...}

    //Accessors
    public int getRowCount() { return rows; }

    public int getColumnCount() { return cols; }

    public void swapRow(int row1, int row2)  1 usage
    {...}

    public void scaleRow(int rowNum ,double k)  1 usage
    {...}

    public void addRow(int targetRow, int sourceRow, double k)  1 usage
    {...}

}
```

Figure 14: Encapsulation in Matrix class

The attributes of the Matrix are protected meaning that they can't be modified outside the class level. This prevents any unforeseen altercations with the size of the matrix. The methods are made public. For example, the accessors allow for information retrieval about the matrix whilst the mutators encapsulate the logic for modifying the matrix.

b. Inheritance

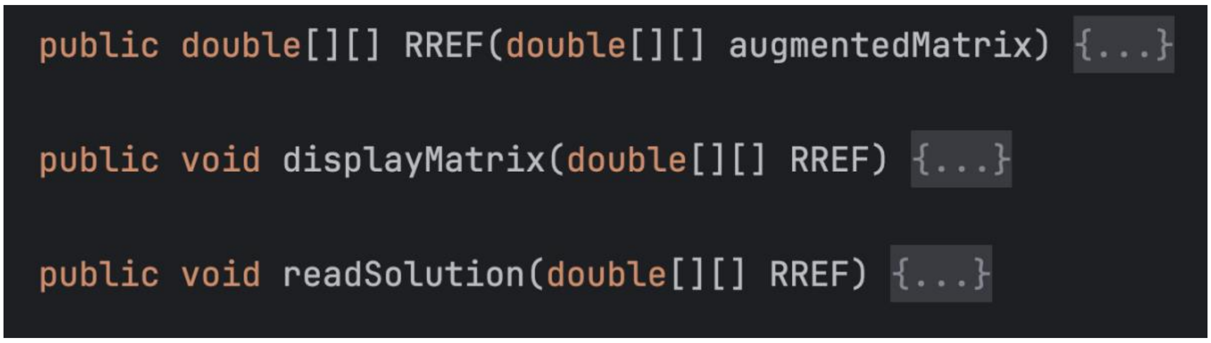


```
class Guassian extends Matrix {  2 usages
    private double[][] augmentedMatrix;  2 usages

    public Guassian(double[][] augmentedMatrix) {  1 usage
        super(augmentedMatrix);
        this.augmentedMatrix = augmentedMatrix;
    }

    public double[][] RREF(double[][] augmentedMatrix) {...}

    public void displayMatrix(double[][] RREF) {...}

    public void readSolution(double[][] RREF) {...}
}
```

Figure 15: Gaussian extends Matrix: Inheritance relationship

Gaussian class extends the Matrix class. This allows the Gaussian class to inherit Matrix methods that are used in the RREF method. This promotes code reusability such as the methods implemented in the Matrix. The super ensures that the Matrix constructor initializes the matrix attributes before any additional initialization int the Gaussian class occurs.

C. Abstraction



```
public double[][] RREF(double[][] augmentedMatrix) {...}

public void displayMatrix(double[][] RREF) {...}

public void readSolution(double[][] RREF) {...}
```

Figure 16: Diagonal 2d array traversal pattern

These methods break down complex tasks into methods that can be implemented quickly and seamless hiding all complexities by encapsulating all RREF operations.

Bibliography

Bretscher, O. (2013). Linear Algebra with Applications (5th ed.). Pearson.

Citation format: GeeksforGeeks. (n.d.). Encapsulation in Java. Retrieved from

https://www.geeksforgeeks.org

CodeHS. (n.d.). Traversing a 2D Array in Java. Retrieved from https://www.codehs.com