ASHESI UNIVERSITY

**Equations In Motion: A Graphical Exploration**

Group (Number) Project

MATH 212: Linear Algebra

Dr. Justice K. Appati

July 26, 2024.

Group Members:

1. Caleb Okwesie Arthur

2. Edem Korbla Anagbah

3. Frances Seyram Fiahagbe

**Solving and Visualizing Systems of Linear Equations In Python**

## Introduction

Systems of linear equations are a cornerstone of numerous fields of study. While they might initially appear as mere collections of mathematical equations and matrices, their applications are extensive and integral to many aspects of daily life when examined through a mathematical lens. These systems can be found in various spheres of study, including engineering, where they are used for structural analysis, circuit design, and control systems; economics, where they model supply and demand; physics, where they are applied in mechanics and electrical circuits; and computer science, where they are used in computer graphics and machine learning. Given their widespread nature, understanding the solution spaces of these systems is essential for addressing real-world problems.

This linear algebra project is dedicated to developing a Python program capable of solving both homogeneous and non-homogeneous systems of linear equations and visualizing their solution sets. By exploring these systems, their solutions, and their graphical representations, our application can provide users with deeper insights into the nature of solutions and their interrelationships.

This report provides a comprehensive overview of our application. It starts by detailing the problem and providing relevant background information. It also elaborates on the methods employed and concludes with the results and their implications.

## Historical Background

Before delving into the intricacies of our applications, it is essential to understand the concept of a system of linear equations. "Mathisfun.com" describes this concept efficiently as a situation when two or more linear equations work together (*Systems of Linear Equations*, n.d.). The objective of such a system is to find the values of these variables that simultaneously satisfy all the equations in the system. Systems of linear equations can be classified into homogeneous and non-homogeneous based on the constant terms on the right-hand side of the equations. Simply, a homogeneous system has all the constant terms (usually depicted by "**b**"**)** equal to zero, while in a non-homogeneous system, at least one of the constant terms is non-zero.

Having realized what a system of linear equations is, it is crucial to understand that the quest to solve systems dates back to ancient civilizations. The Babylonians were the first civilization to develop an organized strategy for solving linear equations, using two linear equations as

representations known as the “method of false position” (Admin, 2023). This method involved first making a guess, evaluating the guess, and then adjusting the guess to make the calculated outcome match the desired result. The Egyptians also used linear equations extensively to calculate dimensions from pyramids and other structures and their volumes, as well as irregular objects like human bodies (Admin, 2023). Later, in ancient China, the text "The Nine Chapters in the Mathematical Art" described methods for solving systems of linear equations using matrix-like arrangements. The Chinese used a technique similar to what we now call the Gaussian elimination method. They would first write the equations in a grid and eliminate the variables by systematically adding or subtracting multiples of rows from each other.

Today, with the advancement of computational tools, solving systems of linear equations has become more streamlined through the use of advanced methods. They include the Gaussian elimination, Gauss-Jordan Elimination, and Cramer's Rule. Each method has its own way of tackling the problem, making use of modern technology to improve both accuracy and efficiency.

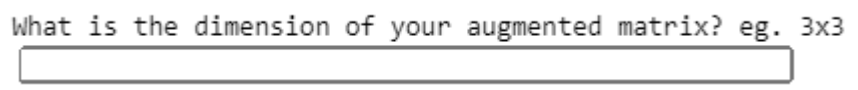
## Methods

Now that the concept of linear algebra has been adequately elaborated on, the methods used in our implementation can be discussed. The core functionality of our application can be summarized into three main methods. First, the input method enables the user to enter the augmented matrix. Second, the solve system method takes this augmented matrix as an argument and solves the system. Finally, the plot solutions method visualizes the various solutions of the system, either in R^2 or R^3. The following sections provide an in-depth exploration of the application's functionality.

**Input_matrix** Method

The inout method comes in two forms allows users to manually enter the elements of the augmented matrix in form-style, based on the size they specify or directly in a Text area.

```
# Finding the dimension and extracting the rows and columns
dimension = input("What is the dimension of your augmented matrix? eg. 3x3\n")
dimension = dimension.lower().replace(" ", "")
index = dimension.find("x")
row = int(dimension[:index])
column = int(dimension[index+1:])
```



To handle the matrix population, the method initially uses symbols as placeholders because SymPy matrices can't be created empty.

```
# Creating a matrix with symbolic placeholders
elements = [sp.Symbol(f'x{i}{j}') for i in range(row) for j in range(column)]
matrix = sp.Matrix(row, column, elements)
```

Users then fill in the actual values for each element through a loop, which continues until the entire matrix is complete.

```
# Populating the matrix
for i in range(row):
    for j in range(column):
        num = input(f"Enter the digit in row {i} and column {j}\n")
        matrix[i, j] = int(num)
```

Besides collecting the augmented matrix, the method also creates the homogeneous matrix. This involves removing the constants (b) from the augmented matrix([A b]) and adding the zero vector. The input method returns both the user-provided augmented matrix and the homogeneous matrix.

```
#Creating the homogeneous matrix
zero_vector = sp.zeros(row, 1)
homo_matrix = matrix[:, :-1]
homo_matrix = homo_matrix.col_insert(homo_matrix.cols, zero_vector)

return matrix, homo_matrix
```

**Solve_Systems** Method

The solve system method takes an augmented matrix as its argument. It first computes the row-reduced echelon form of the matrix and then converts the solution to a NumPy array. It then splits the augmented matrix into the matrix **A** and the constant **b**.

```
matrix = sp.Matrix(augmented_matrix).rref()[0]
matrix = np.array(matrix).astype(np.float64)

# Extract the coefficient matrix and constants
A = matrix[:, :-1]
b = matrix[:, -1]
```

In an attempt to solve the system, the program first checks for inconsistency. It does this by looping through the rows in the matrix, checking if all the entries in a particular row are zero and whether the corresponding entry in the constant b is not zero. If this condition is satisfied for any of the rows in the matrix, then the method returns that the system is inconsistent.

If the system is consistent, then we extract the solutions using the row-reduced echelon form of the matrix. It first loops through the rows and searches for the first non-zero element. This element's column index is considered the pivot column. If a pivot column is found, it's stored in "pivot_col". There is another set that stores the free variables("free_vars"), and if the pivot column is found, it is removed from the set of free variables. The solution for the variable associated with the pivot column is formatted and displayed, and lastly, a message is displayed for all the free variables stored in the "free_vars" list. This particular piece of code was AI generated from ChatGPT.

```
num_rows, num_cols = A.shape
solutions = []
free_vars = set(range(num_cols))
```

```
# Check for inconsistency
for row in range(num_rows):
    if np.all(A[row, :] == 0) and (b[row] != 0):
        return "No solution: Inconsistent system"

# Extract solutions based on the RREF form of the matrix
for row in range(num_rows):
    pivot_col = next((col for col in range(num_cols) if A[row, col] != 0), None)
    if pivot_col is not None:
        free_vars.remove(pivot_col)
        solutions.append(f'x{pivot_col+1} = {b[row]} ' +
                         ' '.join([f'- {A[row,col]}*x{col+1}' for col in free_vars

# Handle free variables
for var in free_vars:
    solutions.append(f'x{var+1} is a free variable')
```

**Plot_Solution** Method

The function takes an augmented matrix as an argument, calls the `solve_systems` method, and assigns the result to a variable. It determines the number of variables by subtracting one from the shape of the augmented matrix (to account for the augmented column). Then, it splits the augmented matrix into matrix A and vector b, where b could be either the zero vector or a user-provided vector.

```
def plot_solution(augmented_matrix):
    #solving the system of the inputted matrix
    solutions = solve_systems(augmented_matrix)

    #Getting the number of variables based on the number of columns
    num_vars = augmented_matrix.shape[1] - 1

    #splitting the matrix into A and B
    A = augmented_matrix[:, :-1]
    b = augmented_matrix[:, -1]
```

The method contains three conditional branches based on the number of variables: two, three, or neither. These branches correspond to vectors in R2 and R3, which are the only dimensions the application can plot. This differentiation is crucial for the program's functionality.

For two variables, the code generates x-values and uses them to compute corresponding y-values for the linear equation. It iterates through each equation, computes the y-values using the line's slope-intercept form, and plots the solutions with appropriate labels.

```
if num_vars == 2:
    # Plotting the lines representing the equations
    x_vals = np.linspace(-10, 10, 400)
    plt.figure()

    for i in range(A.shape[0]):
        y_vals = (b[i] - A[i, 0] * x_vals) / A[i, 1]
        plt.plot(x_vals, y_vals, label=f'Equation {i+1}')

    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.axhline(0, color='black',linewidth=0.5)
    plt.axvline(0, color='black',linewidth=0.5)
    plt.grid(color = 'gray', linestyle = '--', linewidth = 0.5)
    plt.legend()
    plt.title('Solution to the System in R2')
    plt.show()
```

For three variables, The program creates a new 3D plot, generates 400 evenly spaced x-values and y-values between -10 and 10, and uses these to compute corresponding z-values for the linear equation. It loops through each equation, computes the z-values and plots them on the graph.

```
elif num_vars == 3:
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    x_vals = np.linspace(-10, 10, 400)
    y_vals = np.linspace(-10, 10, 400)

    X, Y = np.meshgrid(x_vals, y_vals)

    for i in range(A.shape[0]):
        Z = (b[i] - A[i, 0] * X - A[i, 1] * Y) / A[i, 2]
        ax.plot_surface(X, Y, Z, alpha=0.5, label=f'Equation {i+1}')

    ax.set_xlabel('x1')
    ax.set_ylabel('x2')
    ax.set_zlabel('x3')
    plt.title('Solution to the System in R3')
    plt.show()
```

Otherwise, it displays an error message if the number of variables is not two or three.

This structure ensures that the system can only plot equations in the dimensions it supports, providing clear and accurate visualizations for R2 and R3.

## Results

This part of the report serves to provide some test cases of how the application program works.

### Unique Solution (2D)

An example of a system with a unique solution

$2x_1 + 3x_2 = 8$ $1x_1 + 2x_2 = 5$

Choose input method

Text Area

Enter the augmented matrix [A | b] as shown (NB: Click the empty background to continue):

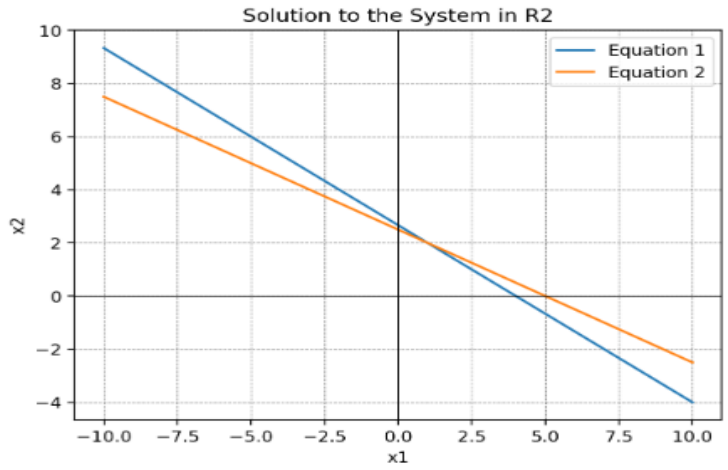
```
2 3 8
1 2 5
```

Solve

**Non-homogeneous System Solution:**

x1 = 1.0

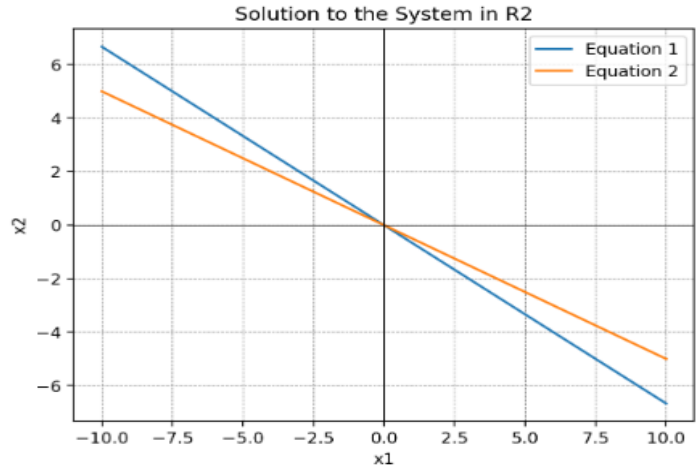
x2 = 2.0




**Homogeneous System Solution:**

x1 = 0.0

x2 = 0.0




**Infinite Solutions (Dependent Equations)**

An example of a system with infinite solutions

$x_1 + x_2 = 2$ $2x_1 + 2x_2 = 4$

Choose input method

Text Area

Enter the augmented matrix [A | b] as shown (NB: Click the empty background to continue):

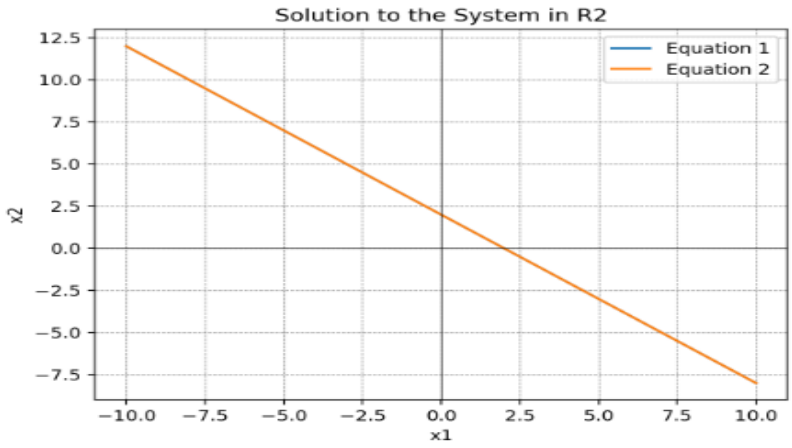
```
1 1 2
2 2 4
```

Solve

**Non-homogeneous System Solution:**

x1 = 2.0 - 1.0*x2

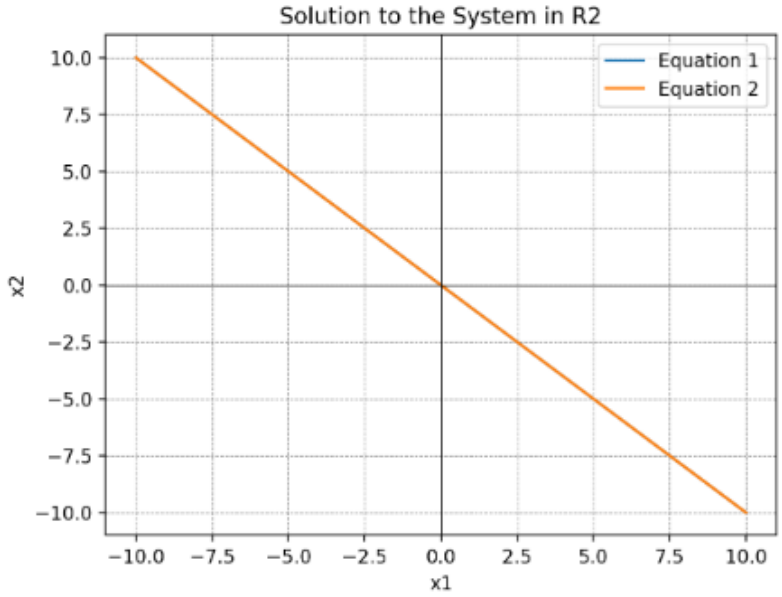
x2 is a free variable




**Homogeneous System Solution:**

x1 = 0.0 - 1.0*x2

x2 is a free variable




### 3D Example with Unique Solution

For a system in three dimensions

$x_1 + x_2 + x_3 = 6$ $0x_1 + x_2 + x_3 = 5$ $x_1 + 0x_2 + x_3 = 4$

Choose input method

Text Area

Enter the augmented matrix [A | b] as shown (NB: Click the empty background to continue):

```
1 1 1 6
0 1 1 5
1 0 1 4
```

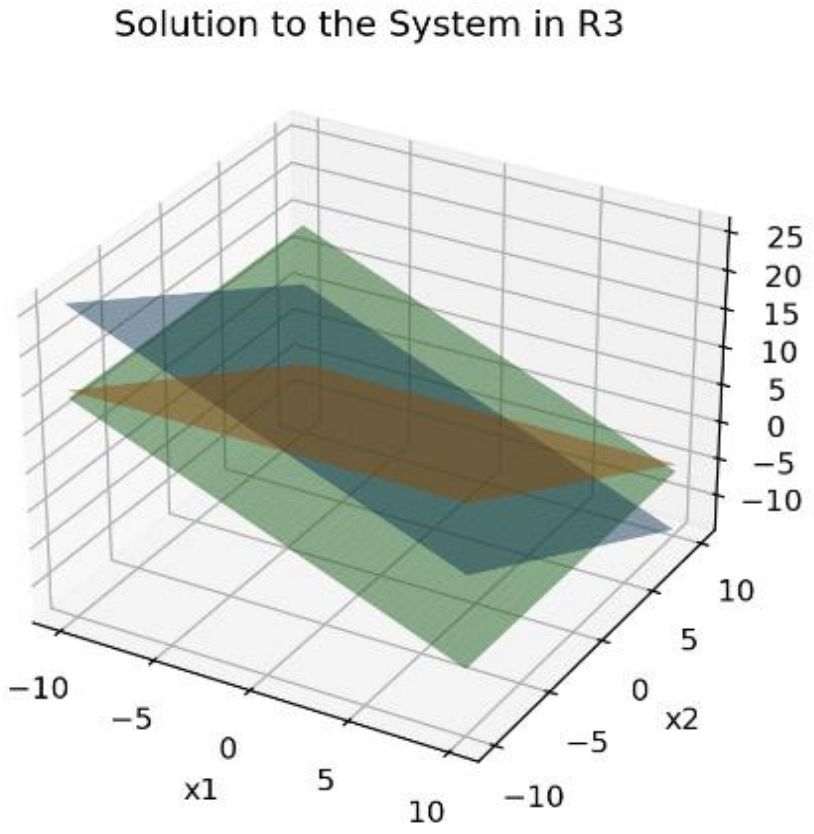
Solve

## Non-homogeneous System Solution:

x1 = 1.0

x2 = 2.0

x3 = 3.0

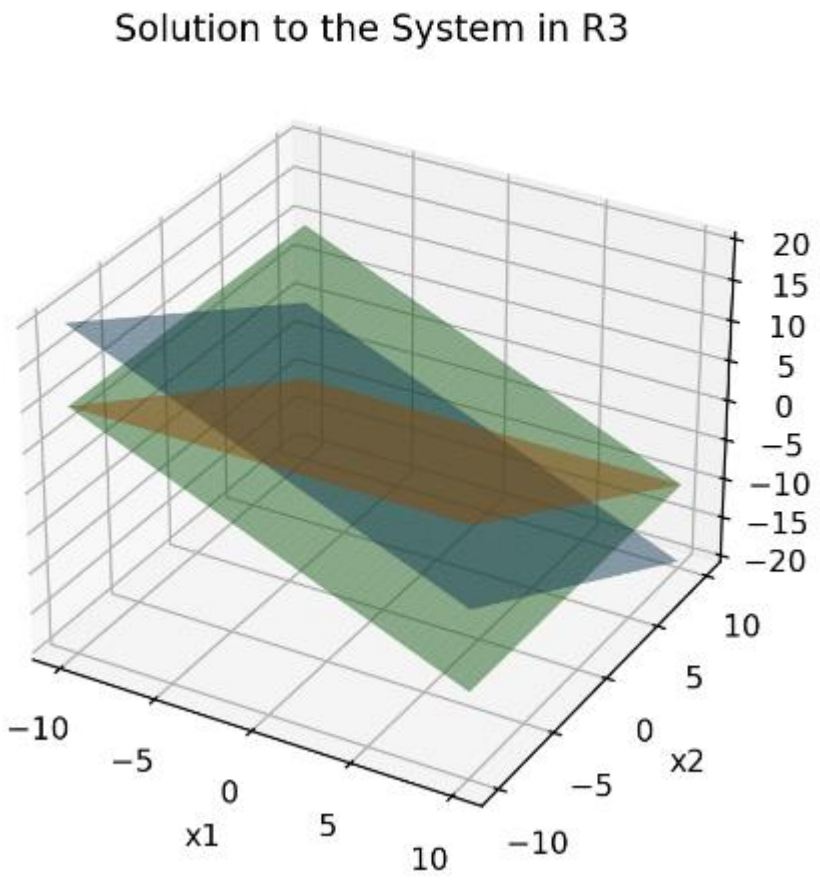


## Homogeneous System Solution:

x1 = 0.0

x2 = 0.0

x3 = 0.0



**3D Example with Infinite Solutions**

An example where the system has infinite solutions (planes intersect along a line)

$x_1 + 2x_2 + 3x_3 = 9$ $2x_1 + 4x_2 + 6x_3 = 18$ $3x_1 + 6x_2 + 9x_3 = 27$

Choose input method

Text Area

Enter the augmented matrix [A | b] as shown (NB: Click the empty background to continue):

```
1 2 3 9
2 4 6 18
3 6 9 27
```

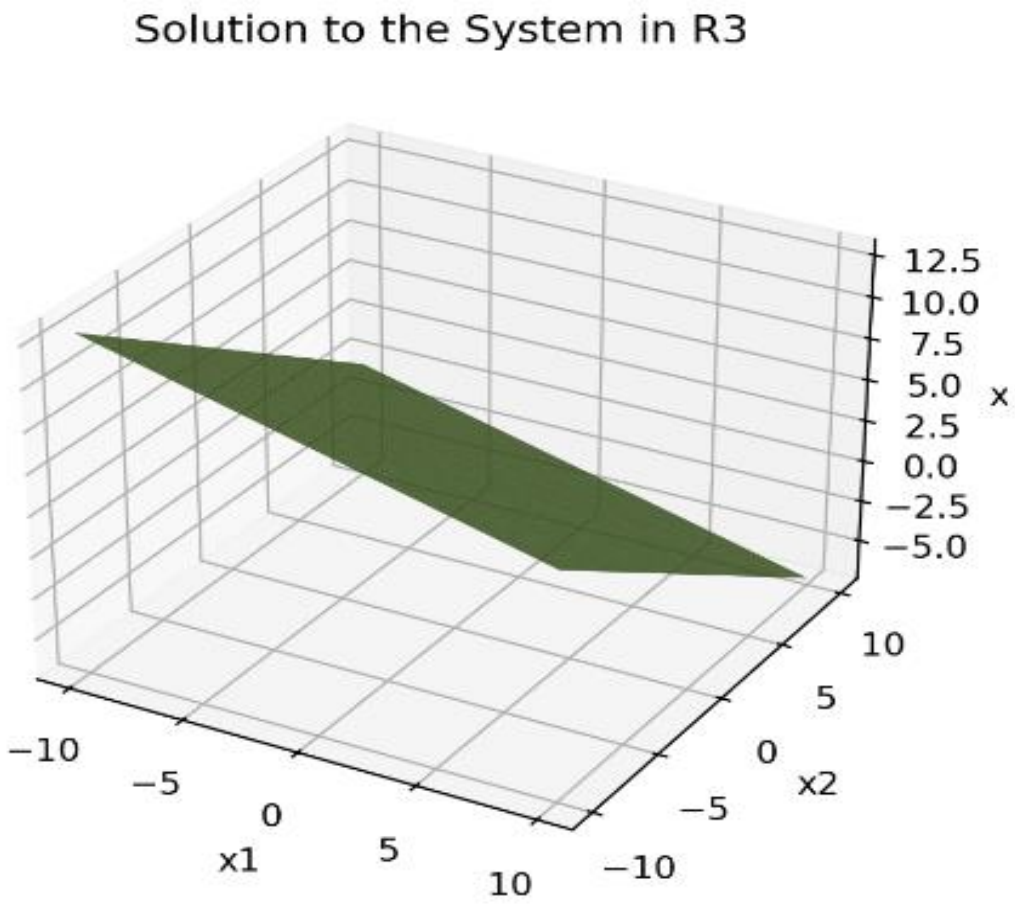
Solve

## Non-homogeneous System Solution:

x1 = 9.0 - 2.0x2 - 3.0x3

x2 is a free variable

x3 is a free variable

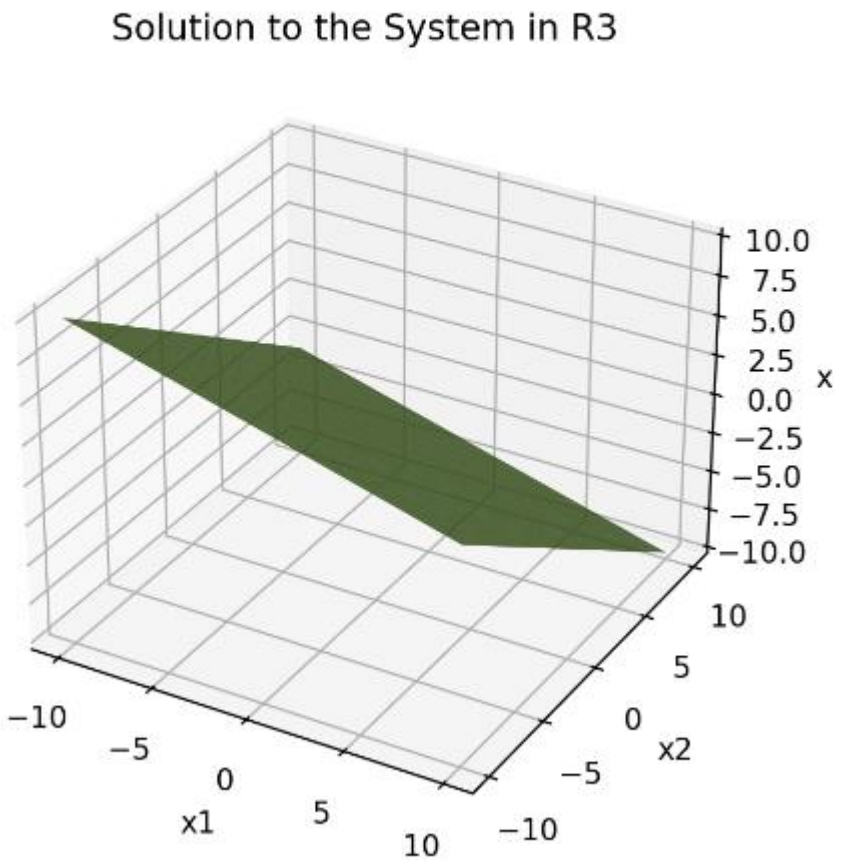


## Homogeneous System Solution:

x1 = 0.0 - 2.0*x2* - *3.0*x3

x2 is a free variable

x3 is a free variable



**No Solution (Inconsistent System)**

An example of a system with no solution

$2x_1 + 3x_2 = 8$ $\qquad$ $4x_1 + 6x_2 = 18$

Choose input method

Text Area

Enter the augmented matrix [A | b] as shown (NB: Click the empty background to continue):

```
2 3 8
4 6 18
```

Solve

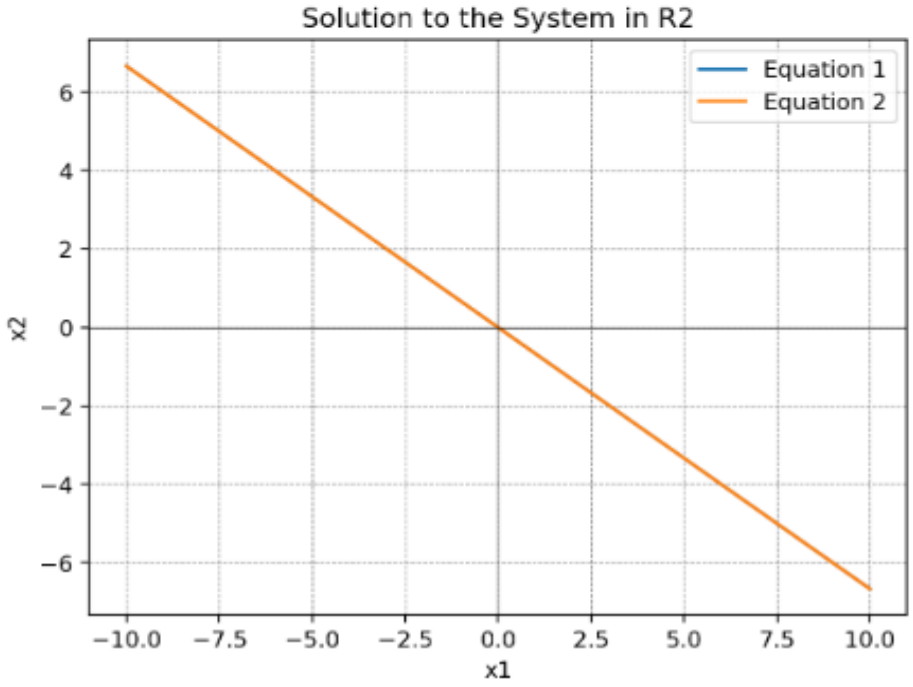
## Non-homogeneous System Solution:

No solution: Inconsistent system

## Homogeneous System Solution:

x1 = 0.0 - 1.5*x2

x2 is a free variable



**Discussion**

After trying the system with 5 different matrices, we made some observations.

For the first system, it has a unique solution. The graphical representation of this system in 2D would showed two intersecting lines, confirming that the solution is where these lines cross. This outcome aligns with our expectations for a system of linear equations with a unique solution in 2D. The unique intersection point represents the specific values of x1 and x2 that satisfy both equations.

For the second system, we observe infinite solutions. The second equation is a multiple of the first, indicating that the two equations are dependent and represent the same line. Graphically, this means that both lines coincide, and there are infinitely many solutions along this line. This confirms the theoretical understanding that when equations are dependent, they represent the same geometrical entity, resulting in infinite solutions.

When comparing the homogeneous and non-homogeneous systems, we noticed distinct differences in their plots. The homogeneous system, where all constant terms are zero, always includes the origin (0,0) as a solution. Its plot passes through this point, creating a notably different visual representation compared to the non-homogeneous system. The non-homogeneous system, with non-zero constant terms, shifts the lines or planes away from the origin, altering the solution space and often resulting in a more diverse range of possible outcomes.

For the third system, we observe a unique solution. In 3D space, this system represents three planes intersecting at a single point. The visualization showed these planes meeting at a single point, which is the unique solution to the system. This matches our expectations for a 3D system with a unique solution where the planes intersect precisely at one point.

For the fourth system, we observe infinite solutions. The second and third equations are multiples of the first, indicating that they represent parallel planes in 3D space. Since these planes intersect along a line, the system has infinitely many solutions. This result is consistent with the understanding that when the planes are parallel, they intersect along a line rather than at a single point.

For the fifth system there is no solution. The lines represented by these system's equations are parallel and do not intersect. This indicates that the system is inconsistent, as there is no point that satisfies both equations simultaneously. This outcome is as expected for parallel lines, which do not share a common solution. In the graph however the lines of on the equations is not visible because the size of the grid thus is not clear how farther the two equations are from each other.

## References

*History of Linear Equations in Math - Mathnotspedia*. (2023, August 21). https://mathnotespedia.in/history-of-linear-equations-in-math/

OpenAi. (2024). *ChatGPT*. Chatgpt.com; OpenAI. https://chatgpt.com

*Systems of Linear Equations*. (2019). Mathsisfun.com. https://www.mathsisfun.com/algebra/systems-linear-equations.html