Mini Project Report: Matrix Rank and Nullity Calulator

Members

1. Nunthatinn Veerapaiboon (Poon) - 01324092
2. Atchariyapat Sirijirakarnjaroen (Beam) - 01324084
3. Nachayada Pattaratichakonkul (May) - 01324073
4. Petch Suwapun (Diamond) - 01324097
5. Thanawin Pattanaphol (Win) - 01324096

Introduction

Matrices are essential mathematical tools used across many fields, including linear algebra, physics, engineering, computer science, and data analysis. Understanding a matrix's key properties—especially its rank and nullity—is vital for solving systems of linear equations, determining whether vectors are linearly independent, and studying linear transformations.

This project focuses on developing a computational tool that can automatically calculate the rank and nullity of any given matrix using the Gaussian elimination method.

In simple terms, the rank of a matrix tells us the number of linearly independent columns (or rows) it contains, which also reflects the dimension of its column or row space. The nullity, on the other hand, represents the dimension of the null space—essentially, all possible solutions to the equation $Ax = 0$. These two values are connected through the Rank-Nullity Theorem, which states that for any m x n matrix ,

Rank + nullity = n

N is the numbers of columns in the matrix

Mathematical background and explanation

1. Span and Linear Combination

The *span* of a set of vectors is the set of all possible *linear combinations* of those vectors.
In other words, for vectors $V_1, V_2, \ldots, V_n \in \mathbb{R}^M$:

$$\text{Span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\} = \{c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n \mid c_1, c_2, \ldots, c_n \in \mathbb{R}\}$$

2. Linear Dependence and Independence

A set of vectors is *linearly dependent* if at least one vector can be expressed as a linear combination of others.

If the only solution to

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n = \mathbf{0}$$

Is $c_1 = c_2 = \cdots = c_n = 0.$, then the set is *linearly independent*.

3. Dimension, Rank, and Nullity

1. Rank:
The rank of a matrix A is the number of linearly independent columns (or rows).
It indicates the *dimension of the column space* (or row space).

2. Nullity:
The nullity of a matrix A is the dimension of its *null space* , the set of all solutions to Ax=0

Rank–Nullity Theorem:
For an m×n matrix A:

$$\text{rank}(A) + \text{nullity}(A) = n$$

where n is the number of columns.

4. Gaussian Elimination

A method of transforming a matrix into a simpler form (REF or RREF) using *elementary row operations*:

1. Swap rows
2. Multiply a row by a nonzero scalar
3. Add/subtract a multiple of one row from another

5. Row Echelon Form (REF) and Reduced Row Echelon Form (RREF)

REF Conditions:

- All nonzero rows are above any rows of zeros.
- Each leading entry (pivot) is to the right of the pivot in the row above.
- All entries below each pivot are zero.

- **RREF Conditions:**
  - Matrix is in REF.
  - Each pivot is 1.
  - Each pivot is the only nonzero entry in its column.

- **Application:**
  - The **number of pivots** = **rank**.
  - The **number of free variables** (columns without pivots) = **nullity**.

Methodology

Overview

This program calculates the **rank** and **nullity** of a matrix using the **Gaussian elimination method**. The idea is to systematically simplify a matrix using basic row operations until it reaches **row echelon form (REF)**. Once in this form, the number of non-zero rows tells us the rank, and the nullity can be found by subtracting that rank from the number of columns.

Algorithm Design

Helper Functions

Before performing elimination, the program defines a few utility functions to make the process cleaner and easier to follow:

- **print_matrix(matrix)**: Displays the matrix in a readable form at each step, allowing us to track progress as the algorithm modifies rows.
- **count_non_empty_rows(matrix)**: Counts how many rows contain at least one non-zero element. This number corresponds to the rank after elimination.
- **swap_rows(matrix, row1, row2)**: Exchanges two rows in the matrix. This is used when the current pivot element is zero, and a non-zero pivot needs to be moved up from below.

Gaussian Elimination

The main logic resides in the **gaussian_elimination()** function, which performs the standard elimination process:

1. **Find a Pivot:**
   For each column, the program looks for the first non-zero entry starting from the current row downward. That element becomes the pivot.
2. **Row Swapping (if needed):**
   If the pivot is not in the current row, the program swaps rows so the pivot moves to the top of the working submatrix.
3. **Normalize the Pivot Row:**
   Once the pivot is identified, the program divides the entire pivot row by its pivot value, making the pivot equal to 1. This simplifies further calculations.
4. **Eliminate Below:**
   The algorithm removes all entries below the pivot by subtracting suitable multiples of the pivot row from the rows underneath, turning those entries into zeros.
5. **Repeat:**
   The pivot position moves diagonally down-right, and the process repeats until the matrix reaches row echelon form

Rank and Nullity Calculation

Once the matrix is in row echelon form:

- **Rank** = number of non-zero rows (using **count_non_empty_rows()**)
- **Nullity** = total number of columns − rank

This is based on the **Rank–Nullity Theorem**, which relates the dimensions of a matrix's column space and null space.

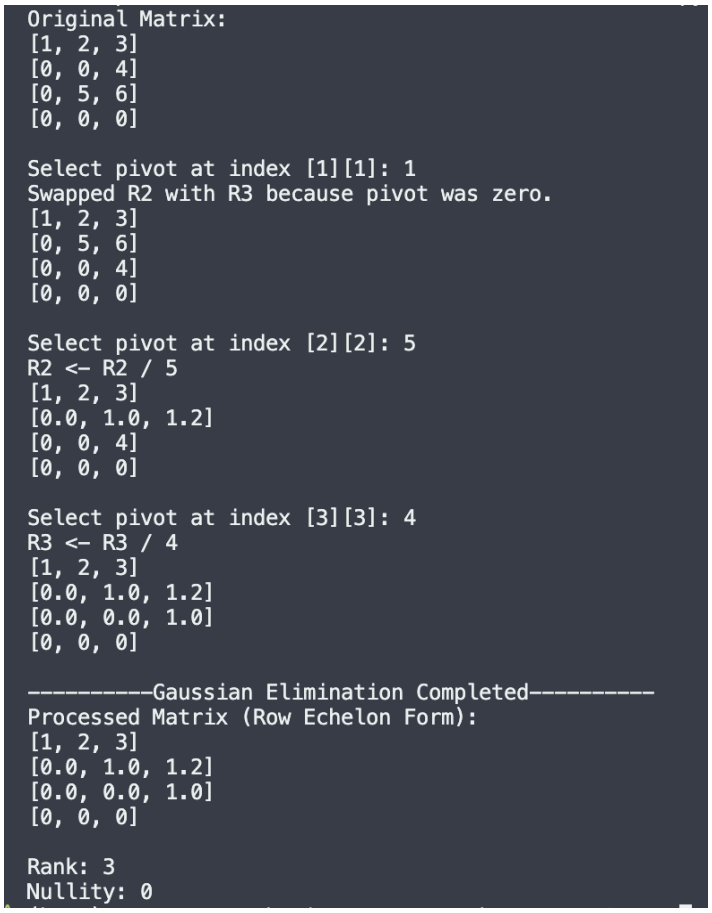
Full Coding Implementation

```python
def print_matrix(matrix):
    for row in matrix:
        print(row)
    print()


def count_non_empty_rows(matrix):
    count = 0
    for row in matrix:
        if any(x != 0 for x in row):
            count += 1
    return count


def swap_rows(matrix, row1, row2):
    matrix[row1], matrix[row2] = matrix[row2], matrix[row1]


def gaussian_elimination(matrix):
    rows = len(matrix)
    cols = len(matrix[0])
    processing_matrix = [row[:] for row in matrix]  # deep copy

    r = 0  # current row for pivot
    c = 0  # current column for pivot

    while r < rows and c < cols:
        # Find pivot in column c at or below row r
        pivot_row = None
        for k in range(r, rows):
            if processing_matrix[k][c] != 0:
                pivot_row = k
                break

        if pivot_row is None:
            # Entire column is zero, move to next column
            print(f"Pivot at column {c+1} is zero, cannot eliminate this column.")
            c += 1
            continue

        # Swap to put pivot at row r
        if pivot_row != r:
            processing_matrix[r], processing_matrix[pivot_row] = processing_matrix[pivot_row], processing_matrix[r]
            print(f"Swapped R{r+1} with R{pivot_row+1} because pivot was zero.")
            print_matrix(processing_matrix)

        cur_pivot = processing_matrix[r][c]
        print(f"Select pivot at index [{r+1}][{c+1}]: {cur_pivot}")

        # Normalize pivot row (optional)
        if cur_pivot != 1:
            print(f"R{r+1} <- R{r+1} / {cur_pivot}")
            for j in range(c, cols):
                processing_matrix[r][j] /= cur_pivot
            print_matrix(processing_matrix)

        # Eliminate rows below
        for k in range(r + 1, rows):
            factor = processing_matrix[k][c]
            if factor == 0:
                continue
            print(f"R{k+1} <- R{k+1} - ({factor}) x R{r+1}")
            for j in range(c, cols):
                processing_matrix[k][j] -= factor * processing_matrix[r][j]
            print_matrix(processing_matrix)

        # Move to next pivot row and next column
        r += 1
        c += 1

    return processing_matrix
```

Results

The result of the program is shown below. The implementation successfully performs **Gaussian elimination** and correctly transforms the input matrix into its **row echelon form (REF)**. Throughout the process, the program prints detailed logs showing each step, including row swaps, normalization, and elimination operations. Finally, it outputs the computed **rank** and **nullity** values. All features work as expected and produce accurate results.

Example Runs



```
Original Matrix:
[1, 2, 1, 0, 3]
[2, 4, 0, 1, 7]
[1, 2, 1, 1, 4]
[0, 0, 1, 1, 2]

Select pivot at index [1][1]: 1
R2 <- R2 - (2) x R1
[1, 2, 1, 0, 3]
[0, 0, -2, 1, 1]
[1, 2, 1, 1, 4]
[0, 0, 1, 1, 2]

R3 <- R3 - (1) x R1
[1, 2, 1, 0, 3]
[0, 0, -2, 1, 1]
[0, 0, 0, 1, 1]
[0, 0, 1, 1, 2]

Pivot at column 2 is zero, cannot eliminate this column.
Select pivot at index [2][3]: -2
R2 <- R2 / -2
[1, 2, 1, 0, 3]
[0, 0, 1.0, -0.5, -0.5]
[0, 0, 0, 1, 1]
[0, 0, 1, 1, 2]

R4 <- R4 - (1) x R2
[1, 2, 1, 0, 3]
[0, 0, 1.0, -0.5, -0.5]
[0, 0, 0, 1, 1]
[0, 0, 0.0, 1.5, 2.5]

Select pivot at index [3][4]: 1
R4 <- R4 - (1.5) x R3
[1, 2, 1, 0, 3]
[0, 0, 1.0, -0.5, -0.5]
[0, 0, 0, 1, 1]
[0, 0, 0.0, 0.0, 1.0]

Select pivot at index [4][5]: 1.0
----------Gaussian Elimination Completed----------
Processed Matrix (Row Echelon Form):
[1, 2, 1, 0, 3]
[0, 0, 1.0, -0.5, -0.5]
[0, 0, 0, 1, 1]
[0, 0, 0.0, 0.0, 1.0]

Rank: 4
Nullity: 1
```

Conclusion and reflection

This project has taught us immensely on mathematical theories and concepts. These concepts are certainly valuable to us in helping to understand more advanced concepts that will be coming our way in the future. These concepts of matrices and linear algebra are a great foundation for our future developments of artificial intelligence and machine learning.

During our project development process, our team has displayed a spectacular display of work ethics and resilience. We chose the idea of a nullity and rank calculator since it is a simple yet very useful tool for learning nullity and rank. We used Python to create our calculator program, though, Python did provide us with a pre-made function in calculating nullity and rank, we decided that it would be best to implement our own function. This ensures that we are able to understand the workings behind the pre-built function that Python provides us.

However, our project is not without obstacles and problems. We experienced a wide range of them, from communication issues, misunderstandings to technical issues; luckily, we managed to get through these issues in our project thanks to our collaborative approach and teamwork.

In summary, we believe that we have done a good job in this project and we hope to continue improving our teamworking skills as we move forward with MAT-210 - Linear Systems.