RANK & NULLITY CALCULATOR

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

THEORETICAL FOUNDATION

1. Span & Linear Combination
2. Rank & nullity
3. Linear Dependence & Independence

SPAN & LINEAR COMBINATION

If you have a set of vectors $\{V_1, V_2, \ldots, V_n\}$, their span is the collection of vectors that can be expressed in the form:

$$\{C_1V_1 + C_2V_2 + \ldots + C_nV_n\}$$

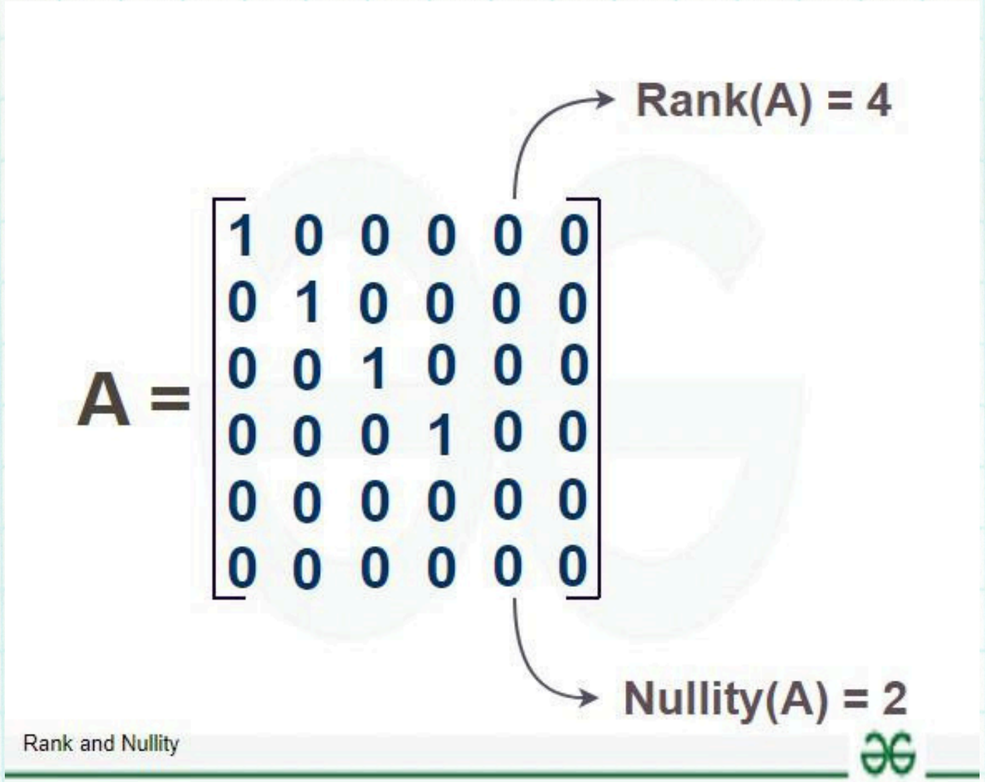
Where c1, c2, . . . ,cn are scalars (real numbers). Essentially, the span of these vectors is the set of all vectors that can be formed by scaling and adding the original vectors.

RANK AND NULLITY

The Rank of a Matrix is a fundamental concept in linear algebra that measures the number of linearly independent rows or columns in a matrix. It essentially determines the dimensionality of the vector space formed by the rows or columns of the matrix.

It helps determine:

- If a system of linear equations has solutions.
- The "usefulness" of rows or columns contributes to the matrix's information.



LINEAR INDEPENDENCE

A set of vectors $\{V_1, V_2, .., V_n\}$ is linearly independent if the vector equation

$$C_1V_1 + C_2V_2 + \ldots + C_nV_n = 0$$

has only the trivial solution $C_1 = C_2 = \ldots = C_n = 0$ The set $\{V_1, V_2, \ldots\ldots, V_n\}$ is linearly dependent otherwise.

LINEAR INDEPENDENCE PART 2

$$\begin{matrix} -1 & 3 & 5 \\ 0 & -2 & 2 \\ 2 & 2 & 2 \end{matrix} \longrightarrow \left[\begin{matrix} -1 & 3 & 5 \\ 0 & -2 & 2 \\ 2 & 2 & 2 \end{matrix} \middle| \begin{matrix} 0 \\ 0 \\ 0 \end{matrix}\right] \xrightarrow{\text{gaussian elimination}} \left[\begin{matrix} 1 & -3 & -5 \\ 0 & 1 & -1 \\ 0 & 0 & -1 \end{matrix} \middle| \begin{matrix} 0 \\ 0 \\ 0 \end{matrix}\right]$$

Default matrix **Augmented matrix** **Row echelon form**

$$\left.\begin{aligned} A_1 - 3A_2 + 5A_3 &= 0 \\ 0A_1 + 1A_2 - 1A_3 &= 0 \\ 0A_1 + 0A_2 - 1A_3 &= 0 \end{aligned}\right\} \quad A_1 = A_2 = A_3 = 0$$

Which mean that this matrix is linearly independent.

Therefore, this matrix has a full rank of 3 and nullity of 0

IMPLEMENTATION

Normal way: Gaussian Elimination & REF

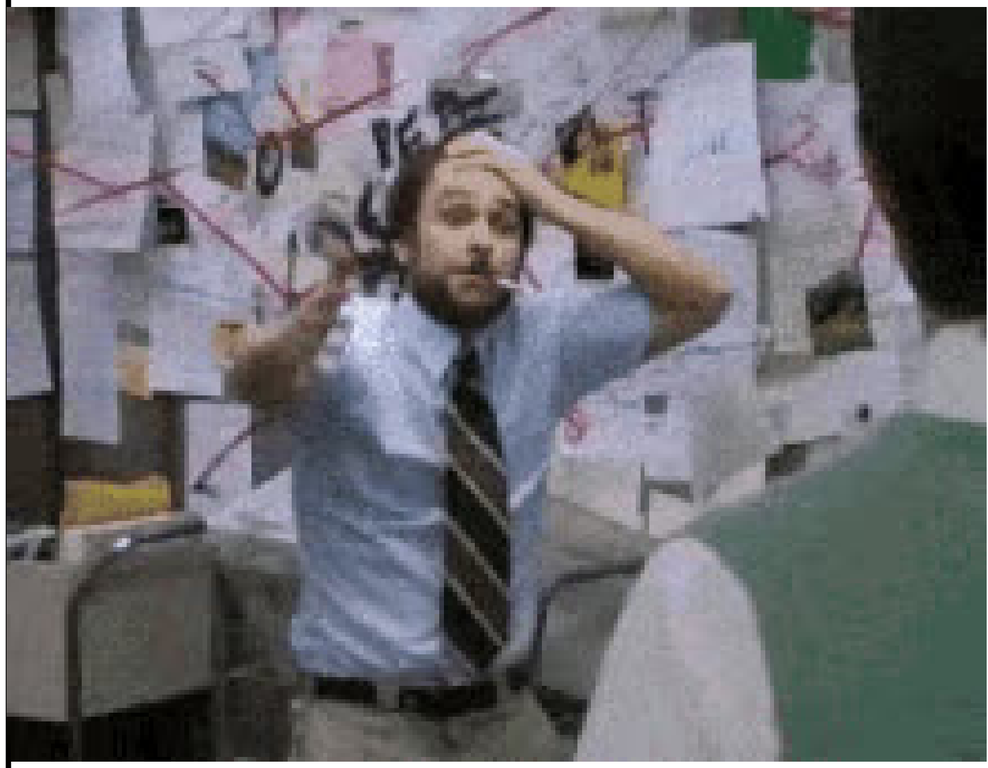
```python
def gaussian_elimination(matrix):
    rows = len(matrix)
    cols = len(matrix[0])
    processing_matrix = [row[:] for row in matrix]  # deep copy

    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
```

IMPLEMENTATION OUTPUT

```
Original Matrix:
[1, 2, 3]
[0, 0, 4]
[0, 5, 6]
[0, 0, 0]

Select pivot at index [1][1]: 1
Swapped R2 with R3 because pivot was zero.
[1, 2, 3]
[0, 5, 6]
[0, 0, 4]
[0, 0, 0]

Select pivot at index [2][2]: 5
R2 <- R2 / 5
[1, 2, 3]
[0.0, 1.0, 1.2]
[0, 0, 4]
[0, 0, 0]

Select pivot at index [3][3]: 4
R3 <- R3 / 4
[1, 2, 3]
[0.0, 1.0, 1.2]
[0.0, 0.0, 1.0]
[0, 0, 0]

----------Gaussian Elimination Completed----------
Processed Matrix (Row Echelon Form):
[1, 2, 3]
[0.0, 1.0, 1.2]
[0.0, 0.0, 1.0]
[0, 0, 0]

Rank: 3
Nullity: 0
```

```
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
```

```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



def main():
    matrix = [
        [1, 2, 1, 0, 3],
        [2, 4, 0, 1, 7],
        [1, 2, 1, 1, 4],
        [0, 0, 1, 1, 2]
    ]



    print("Original Matrix:")
    print_matrix(matrix)

    processed_matrix = gaussian_elimination(matrix)

    rank = count_non_empty_rows(processed_matrix)
    nullity = len(matrix[0]) - rank

    print(f"----------Gaussian Elimination Completed----------")
    print("Processed Matrix (Row Echelon Form):")
    print_matrix(processed_matrix)
    print(f"Rank: {rank}")
    print(f"Nullity: {nullity}")



if __name__ == "__main__":
    main()
```



Reality Check

```python
rank = np.linalg.matrix_rank(A)
```

IMPLEMENTATION ???

Easy way: Use a function in **numpy** library

- numpy.linalg.matrix_rank()

```
def matrix_rank_and_nullity(matrix):
    """
    Calculate the rank and nullity of a matrix.
    :param matrix: list of lists or numpy array
    :return: (rank, nullity)
    """
    A = np.array(matrix, dtype=float)
    rank = np.linalg.matrix_rank(A)
    nullity = A.shape[1] - rank  # number of columns - rank
    return rank, nullity
```

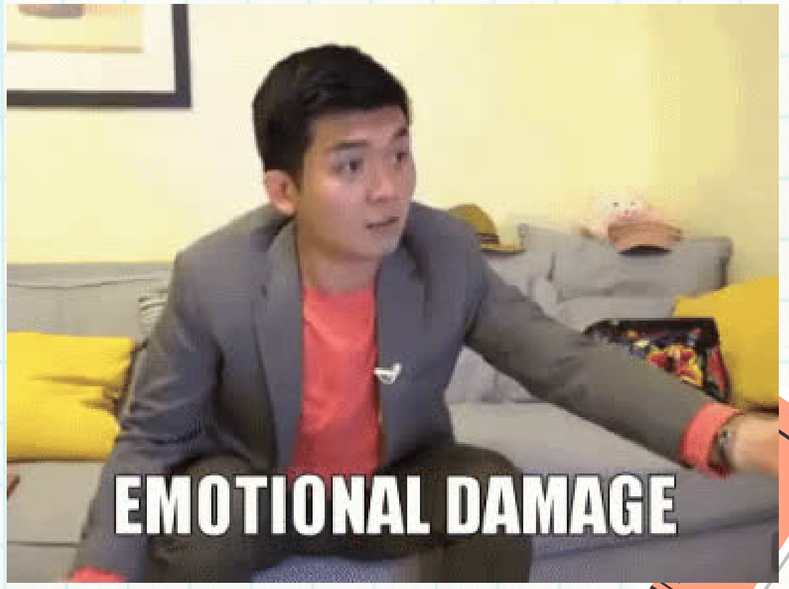
LESSONS LEARNED

Key takeaways

We got deeper understand on basic of matrices, e.g. nullity, rank, linearly dependent.

Challenges

It's sometimes complicated to implement the mathematical method in programming (step by step)



Q & A