BÁO CÁO

LAB 1 - VECTOR VÀ HỆ PHƯƠNG PHƯƠNG TRÌNH TUYẾN TÍNH TRONG PYTHON

1. Kiểm tra ma trận bậc thang

a. Ý tưởng:

1. Hàm duyệt qua từng hàng của ma trận.
2. Trong mỗi hàng, hàm tìm điểm khác không đầu tiên (*pivot*) bằng cách kiểm tra từng phần tử từ trái sang phải.
3. Nếu một *pivot* được tìm thấy:
   - Nếu *pivot* này nằm bên trái hoặc trùng với *pivot* của hàng trước đó, ma trận không ở dạng **echelon** và hàm trả về False.
   - Nếu *pivot* đúng vị trí, hàm tiếp tục kiểm tra các phần tử dưới *pivot* trong cùng cột. Nếu có phần tử nào khác không, ma trận cũng không ở dạng **echelon** và hàm trả về False.
   - Nếu không có phần tử khác không dưới pivot, hàm tiếp tục kiểm tra hàng tiếp theo.
4. Nếu không tìm thấy *pivot* trong hàng hiện tại, hàm kiểm tra các phần tử còn lại bên phải của hàng. Nếu bất kỳ phần tử nào khác không, ma trận cũng không ở dạng **echelon** và hàm trả về False.
5. Nếu không có phần tử khác không trong hàng và không tìm thấy *pivot*, hàm tiếp tục kiểm tra hàng tiếp theo.
   - Nếu qua tất cả các bước trên mà không có điểm cơ bản không hợp lệ nào được tìm thấy, ma trận được xem là ở dạng **echelon** và hàm trả về True.

b. Độ phức tạp: O(n*m), trong đó n là số hàng của ma trận và m là số cột của ma trận.

c. Mã nguồn:

```python
def is_echelon(A):
    num_row, num_col = len(A) , len(A[0])
    prev_pivot_col = -1

    for i in range(num_row):
        found_pivot = False
        for j in range(num_col):
            if not is_zero(A[i][j]):
                if j <= prev_pivot_col:
                    return False
                for k in range(i+1,num_row):
                    if not is_zero(A[k][j]):
                        return False
                prev_pivot_col = j
                found_pivot = True
                break

        if not found_pivot:
            for k in range(j + 1, num_col):
                if not is_zero(A[i][k]):
                    return False
    return True
```

2. Tạo ma trận bậc thang – Sử dụng phép khử Gauss

a. Ý tưởng + Mã nguồn:

Thực hiện theo các bước trong phép Khử Gauss.

Bước 1: Xác định cột trái nhất không chứa toán số 0 với hàm **is_column_zero()**

```python
def is_column_zero(col_index):
        i = col_index
        for i in range(m):
            if not is_zero(A[i][col_index]):
                return False
        return True
```

Bước 2: Đổi chỗ hai dòng (nếu cần thiết), để đưa số hạng khác 0 nào đó ở dưới về đầu cột nhận được ở bước 1. Ở bước này, , chia thành 2 hàm **swap_rows(int row_index)** và **swap_rows_3(int row_index, int col_index)** để xử lí cho 2 trường hợp khác nhau.

- Hàm **swap_rows(int row_index):** Hàm này tìm hàng đầu tiên trong phạm vi từ row_index + 1 đến m (số hàng của ma trận) mà không phải là hàng 0 và có phần tử đầu tiên khác 0 và không khác 1 (do bạn đang xác định một hệ số khác 1 trong quá trình triệt tiêu).

```
def swap_rows(row_index):
        for i in range (row_index + 1,m):
            if not is_zero(A[i][row_index]) and is_one(A[i][row_index]):
                temp = A[row_index]
                A[row_index] = A[i]
                A[i] = temp
                break
```

- Hàm **swap_rows_3(int row_index, int col_index):** Hàm này tìm hàng và cột đầu tiên (bắt đầu từ vị trí row_index + 1 và col_index) mà không phải là 0 và nằm bên phải cột hiện tại của ma trận. Dùng để swap các dòng có nhiều **sub_column_zero** kế tiếp nhau (các cột con có giá trị 0 liên tiếp nhau).

```
def swap_rows_3(row_index, col_index):
        best_row = row_index
        best_col = n

        for i in range(row_index + 1, m):
            for j in range(col_index, n):
                if not is_zero(A[i][j]) and j < best_col:
                    best_row = i
                    best_col = j
                    break

        if best_row != row_index:
            temp_row = A[row_index]
            A[row_index] = A[best_row]
            A[best_row] = temp_row
```

Bước 3: Tạo **số dẫn đầu 1 – leading 1** (*Bước này tùy chọn*) bằng việc nhân dòng chứa nó với **1/a** (a != 0 là số hạng đầu cột nhận được ở Bước 2).

```
def make_leading_one(row_index, col_index=None):
        if col_index is None:
            col_index = row_index
        leading_coefficient = A[row_index][col_index]
        if leading_coefficient != 0 and not is_one(leading_coefficient):
            scale_factor = 1 / leading_coefficient
            A[row_index] = [element * scale_factor for element in A[row_index]]
```

Bước 5: Che dòng đầu đã làm xong. Lặp lại các bước cho đến khi được ma trận bậc thang. Ở bước này, ta sẽ chia bài toán ra thành 3 trường hợp để giải bài toán tối ưu.

```
      #step5: Main loop for Gauss Elimination

    for i in range(min(m, n)):
        if not is_column_zero(i) and not is_sub_column_zero(i,i):
            swap_rows(i)
            if leading1:
                make_leading_one(i)
            eliminate_below(i)
        elif is_sub_column_zero(i,i) :
            if i+1<n and not is_sub_column_zero(i+1,i):
                swap_rows(i - 1)
                if leading1:
                    make_leading_one(i, i+1)
                eliminate_below(i, i+1)
            elif i+1<n and is_sub_column_zero(i+1,i):
                next_column = i + 1
                while next_column<n and is_sub_column_zero(next_column, i) :
                    next_column += 1

                if i+next_column-2<n:
                    swap_rows_3(i,i+next_column-2)
                    if leading1:
                        make_leading_one(i, i+next_column-2)
                    eliminate_below(i, i+next_column-2)
```

Trường hợp 1: Cột hiện tại có phần tử chính không bằng 0

- Hoán vị hàng **(swap_rows(i))** nếu cần.
- Chuẩn hóa phần tử chính thành 1 **(make_leading_one(i)).**

- Loại bỏ các phần tử bên dưới phần tử chính **(eliminate_below(i)).**

Trường hợp 2: Phần tử chính hiện tại bằng 0, nhưng các cột tiếp theo có phần tử chính không bằng 0 bắt đầu từ dòng đang xét trở xuống.

- Hoán vị hàng **(swap_rows(i - 1)).**
- Chuẩn hóa phần tử chính thành 1 **(make_leading_one(i, i + 1)).**
- Loại bỏ các phần tử bên dưới phần tử chính **(eliminate_below(i, i + 1)).**

Trường hợp 3: Cả phần tử chính hiện tại và phần tử chính của các cột tiếp theo bắt đầu từ dòng đang xét trở xuống tiếp theo đều bằng 0.

- Tìm cột tiếp theo có phần tử không bằng **0 (next_column).**
- Hoán vị hàng với hàng tốt nhất **(swap_rows_3(i, i + next_column - 2)).**
- Chuẩn hóa phần tử chính thành 1 **(make_leading_one(i, i + next_column - 2)).**
- Loại bỏ các phần tử bên dưới phần tử chính **(eliminate_below(i, i + next_column - 2)).**

b. Mã nguồn:

```
# Gauss Elimination Function
def Gauss_elimination(A, leading1):
    m, n =len(A), len(A[0])

    #Step1: Check if the leftmost column contains all zeros
    def is_column_zero(col_index):
    def is_sub_column_zero(col_index, row_index):

    #Step2: Swap rows if it !=0;
    def swap_rows(row_index):
    def swap_rows_3(row_index, col_index):

    #Step3: Make leading 1:
    def make_leading_one(row_index, col_index=None):

    #Step4: Make elements below leading term zero
    def eliminate_below(row_index, col_index=None):

    #step5: Main loop for Gauss Elimination

    return A
```

3. Phép thế ngược

a. Ý tưởng:

1. **Đảm bảo đủ số hàng**: Đếm và thêm các vector zero vào ma trận `R` nếu số hàng hiện tại ít hơn `n - 1` (số cột trừ đi 1). Đưa ma trận về dạng n*n trước thi thực hiện Back Substitution.
    - **count_zero_vectors_needed(R)**: Hàm này đếm số vector zero cần thêm vào ma trận `R`. Nếu số hàng ít hơn số cột trừ đi 1, nó trả về số lượng vector zero cần thêm.

```
def count_zero_vectors_needed(R):
        num_rows = len(R)
        num_cols = len(R[0])
        return max(0, num_cols - 1 - num_rows)
```

    - **add_zero_vectors(R)**: Hàm này thêm các vector zero vào ma trận `R` nếu cần thiết. Nó tạo một bản sao của `R`, sau đó thêm các vector zero vào cuối ma trận cho đến khi đủ số lượng hàng.

```
def add_zero_vectors(R):
        num_cols = len(R[0])
        num_zero_vectors_needed = count_zero_vectors_needed(R)

        if num_zero_vectors_needed == 0:
            return R

        # Add zero vectors
        R2 = [row[:] for row in R]  # Copy R to R2
        for _ in range(num_zero_vectors_needed):
            zero_vector = [0] * num_cols
            R2.append(zero_vector)

        return R2
```

2. **Back Substitution**: Giải hệ phương trình tuyến tính từ dưới lên trên (back substitution) để tìm nghiệm của hệ.
    - Khởi tạo danh sách Solutions với giá trị None cho mỗi biến.
    - Duyệt từ dưới lên trên ma trận:
        - **Kiểm tra hàng zero**: Nếu hàng hiện tại là vector zero (tất cả phần tử đều là zero ngoại trừ phần tử cuối cùng), và phần tử cuối cùng không phải zero, hệ vô nghiệm. Nếu hàng là vector zero hoàn toàn, gán biến tự do (free variable).

```
# Check if the row has all zeroes except the last element
        if all(is_zero(row[j]) for j in range(n - 1)):
            if not is_zero(row[-1]):
                return "No solution."
            # Assign free variable
            free_var = None
            for j in range(n - 1):
                if solutions[j] is None:
                    free_var = j
                    break
            if free_var is not None:
                solutions[free_var] = f"x{free_var + 1}"
```

- **Tìm cột pivot**: Xác định cột pivot (phần tử đầu tiên không phải zero trong hàng hiện tại).

```
# Find the pivot column
        pivot_col = None
        for j in range(n - 1):
            if not is_zero(row[j]):
                pivot_col = j
                break
```

- **Tạo phương trình**: Dựa trên giá trị của các phần tử trong hàng và giá trị của các biến đã biết, tạo phương trình cho biến pivot. Nếu biến đã biết, trừ đi giá trị tương ứng từ phần tử cuối cùng của hàng.
- **Gán giá trị cho biến**: Nếu phương trình chứa các biến tự do, gán giá trị cho biến pivot bằng phương trình. Nếu không, tính trực tiếp giá trị của biến pivot.

```
if pivot_col is not None:
            rhs = row[-1]
            equation = []
            for j in range(pivot_col + 1, n - 1):
                if not is_zero(row[j]):
                    if solutions[j] is not None:
                        if isinstance(solutions[j], (int, float, Fraction)):
                            rhs -= row[j] * solutions[j]
                        else:
```

```
                            equation.append(f"({-row[j]})*{solutions[j]}")
                    else:
                        equation.append(f"({-row[j]})*x{j+1}")
            if equation:
                equation_str = " - ".join(equation)
                solutions[pivot_col] = f"{rhs} - {equation_str}"
            else:
                solutions[pivot_col] = rhs / row[pivot_col]
        else:
            free_var = None
            for j in range(n - 1):
                if solutions[j] is None:
                    free_var = j
                    break
            if free_var is not None:
                solutions[free_var] = f"x{free_var + 1}"
```

- Làm sạch nghiệm: Loại bỏ các thành phần zero trong phương trình nghiệm để hiển thị kết quả gọn gàng hơn.

```
def clean_solution(sol):
        if isinstance(sol, str):
            terms = sol.split(" - ")
            cleaned_terms = [term for term in terms if not any(is_zero(float(f))
for f in term.split('*') if f.replace('.', '', 1).isdigit())]
            return " - ".join(cleaned_terms)
        return str(sol)
```

b. Mã nguồn:

```
def back_substitution(R):

    # Function to count the number of zero vectors needed
    def count_zero_vectors_needed(R):

    # Function to add zero vectors if needed
    def add_zero_vectors(R):

     # Count zero vectors needed and add them if necessary
    R = add_zero_vectors(R)
    m, n = len(R), len(R[0])
    solutions = [None] * (n - 1)

    for i in range(m - 1, -1, -1):
```

```
        row = R[i]
        # Check if the row has all zeroes except the last element
            #code
        # Find the pivot column
            #code
        # Assign value to variable
            #code

    # Convert solutions to string and filter out zero terms
    def clean_solution(sol):

    solutions_str = [clean_solution(sol) if sol is not None else "0" for sol in
solutions]

    return solutions_str
```

-HẾT-