BÀI THỰC HÀNH LAB 2

Giảng viên: TS.Dương Việt Hằng

Trợ giảng: Trần Hà Sơn

Ngày 9 tháng 4 năm 2023

Mục tiêu:

- Nắm vững được Python để viết được các đoạn lệnh.
- Nâng cao kỹ năng lập trình cùng với tư duy toán học.
- Nắm vững định nghĩa ma trận và các phép toán trên ma trận, lập trình được các phép toán trên ma trận.

1 Thực hành một số phép toán ma trận với Python

1.1 Khai báo ma trận

Chúng ta có thể sử dụng đồng thời list và NumPy để thực hiện các phép toán trên ma trận. Trong đó, sử dụng list để thực hiện các phép toán bằng code thủ công, trong khi đó NumPy sẽ cho các hàm có sẵn trong thư viện khiến cho các thao tác trở nên đơn giản hơn.

Ví dụ 1. *Cho ma trận* $A = \begin{bmatrix} 1 & 1.5 & -1.2 \\ 2 & 3.7 & 8 \\ 3.5 & 2.5 & 4 \end{bmatrix}$.

a) Biểu diễn ma trận A bằng list như sau:

```
A=[[1,1.5,-1.2],[2,3.7,8],[3.5,2.5,4]]
```

b) Biểu diễn ma trận A bằng NumPy như sau:
Thực hiện khai báo:

```
import numpy as np
```

Sau đó nhập lệnh:

```
A_np=np.array(A)
```

1.2 Các phép toán cơ bản trên ma trận

a) **Nhân một số với ma trận**

Trên list, để thực hiện nhân một số với ma trận, ta xây dựng hàm $multiplyScalarMatrix(scalar, A)$ với:

Input

- scalar: hằng số nhân với ma trận (thực hoặc phức).
- A: ma trận cần tính.

Output

- Ma trận tích $scalar \times A$.

Chương trình như sau:

```
def multiplyScalarMatrix(scalar, A):
    return [[scalar * a for a in a_row] for a_row in A]
# Chay voi ma tran A
A = [[1, 1.5, -1.2],
[2, 3.7, 8],
[3.5, 2.5, 4]]
multiplyScalarMatrix(2, A)
```

Kết quả là:

```
[[2, 3.0, -2.4], [4, 7.4, 16], [7.0, 5.0, 8]]
```

b) **Cộng hai ma trận**

Chương trình sau thực hiện cộng hai ma trận trên list, kết quả xuất ra ma trận là tổng hai ma trận nhập vào từ bàn phím.

```
def CongMaTran(A, B):
     #Khoi tao ma tran ket qua
    C = [[0 for _ in range(len(A[0]))] for _ in range(len(A))]
    #Cong hai ma tran
    m=len(A)#Cho biet so dong cua A
    n=len(A[0])#Cho biet so cot cua A
    for i in range(m):
         for j in range(n):
              C[i][j] = A[i][j] + B[i][j]
    return C
```

Chạy thử với hai ma trận:

```
A=[[1,1.5,-1.2],[2,3.7,8],[3.5,2.5,4]]
B= [[2, 1.5, 1.2],[3, 7, -9.5],[5.5, 2, 4]]
CongMaTran(A,B)
```

Kết quả là

```
[[3, 3.0, 0.0], [5, 10.7, -1.5], [9.0, 4.5, 8]]
```

Thực hiện trên NumPy.

```
A=[[1,1.5,-1.2],[2,3.7,8],[3.5,2.5,4]]
B= [[2, 1.5, 1.2],[3, 7, -9.5],[5.5, 2, 4]]
A_np=np.array(A)
B_np=np.array(B)
A+B
```

Kết quả là

```
array([[ 3. ,  3. ,  0. ],
[ 5. , 10.7, -1.5],
[ 9. ,  4.5,  8. ]])
```

c) **Nhân hai ma trận**

Chương trình sau thực hiện cộng hai ma trận trên list, kết quả xuất ra ma trận là tích hai ma trận nhập vào từ bàn phím nếu hai ma trận nhân được với nhau, ngược lại sẽ thông báo là hai ma trận không phù hợp.

```
def NhanMaTran(A, B):
    #Khoi tao ma tran ket qua
    C = [[0 for _ in range(len(B[0]))] for _ in range(len(A))]
    #Nhan hai ma tran
    m=len(A)#Cho biet so dong cua A
    n=len(A[0])#Cho biet so cot cua A
    p=len(B[0])# Cho biet so cot cua B
    if(n==len(B)):
        for i in range(m):
            for j in range(n):
                total =0
                for k in range(p):
                    total=total+A[i][k]*B[k][j]
                C[i][j] = total
        return C
    print("Hai ma tran khong phu hop")
```

Chạy thử với hai ma trận A, B như sau:

```
A=[[1, 2,4]]
B=[[1,2,3],[2,2,2],[1,1,1]]
NhanMaTran(A, B)
```

Kết quả là

```
[[9, 10, 11]]
```

Trên NumPy, để thực hiện nhân hai ma trận, ta dùng phép tính @.

```
A=[[1, 2,4]]
B=[[1,2,3],[2,2,2],[1,1,1]]
A_np=np.array(A)
B_np=np.array(B)
A_np @ B_np
```

Kết quả là

```
array([[ 9, 10, 11]])
```

d) **Chuyển vị ma trận**
Thực hiên trên list, chương trình sau nhận một ma trận A và trả về ma trận chuyển vị của ma trận A.

```
def transpose(A):
      #Khoi tao ma tran ket qua
      AT = [[0 for _ in range(len(A))] for _ in range(len(A[0]))]
      m=len(A)#Cho biet so dong cua A
      n=len(A[0])#Cho biet so cot cua A
      for i in range(m):
            for j in range(n):
                  AT[j][i]=A[i][j]
        return AT
```

Chạy thử với ma trận $A = \begin{bmatrix} 1 & 2 & 3 \\ 6 & 7 & 8 \end{bmatrix}$.

```
A=[[1,2,3],[6,7,8]]
print(transpose(A))
```

Kết quả là

```
[[1, 6], [2, 7], [3, 8]]
```

Sử dụng hàm của thư viện NumPy, để tìm ma trận chuyển vị của ma trận A, ta sử dụng hàm $np.transpose(\cdot)$.

```
print(np.transpose(A))
```

Kết quả là

```
[[1 6]
[2 7]
[3 8]]
```

2 Đồ án

Bài 1. Sinh viên viết hàm **innerproduct**$(v1, v2)$, trong đó:

- Input: $v1, v2$ là các vector thuộc $\mathbb{R}^n$.
- Output: Giá trị $v1 \cdot v2$ là tích vô hướng của hai vector.

Lưu ý: Sinh viên không được dùng các hàm có sẵn trong thư viện để tính tích vô hướng của hai vector.

Bài 2. Sinh viên viết hàm **QR_factorization(A)**, trong đó:

- Input: A là ma trận có m dòng, n cột với các phần tử là các số thực.
- Output: Ma trận Q và ma trận R.

Lưu ý: Sinh viên không được dùng các hàm có sẵn của thư viện để phân tích ma trận A thành tích QR.

2.1 Quy định bài nộp

- Thực hiện toàn bộ bài làm trên một tập tin Python (.py) hoặc một tập tin Jupyter Notebook (.ipynb).
- Nộp tập tin **MSSV.zip** được nén từ thư mục **MSSV** chứa các tập tin sau:
  1. Báo cáo toàn bộ bài làm: MSSV.pdf.
     Nội dung báo cáo gồm có: Thông tin cá nhân (họ và tên, mã số sinh viên); ý tưởng thực hiện, mô tả các hàm.
  2. Mã nguồn: MSSV.py

2.2 Quy định chấm bài

Những trường hợp sau sẽ bị **0** điểm toàn bộ đồ án:

- Nộp sai quy định.
- Không có báo cáo.
- Thực thi mã nguồn báo lỗi.

LƯU Ý: SAO CHÉP BÀI LÀM CỦA NHAU SẼ BỊ 0 ĐIỂM TOÀN BỘ PHẦN THỰC HÀNH.