TP : Introduction au Calcul Matriciel en Python

ECH-CHAMMAKHY Mohammed

Introduction

Ce TP a pour objectif de vous initier à la manipulation des matrices et aux opérations matricielles en utilisant Python. Les matrices sont des objets mathématiques fondamentaux en algèbre linéaire, et elles sont largement utilisées en mathématiques, en physique, en informatique et dans de nombreux autres domaines. Python, avec la bibliothèque `numpy`, est un outil puissant pour effectuer des calculs matriciels de manière efficace.

Dans ce TP, vous allez apprendre à :

— Créer des matrices en Python.
— Effectuer des opérations de base sur les matrices (addition, multiplication, transposition...).
— Résoudre un système linéaire en python.

Instructions

— Le compte rendu doit être sous forme d'un fichier python contenant la réponse à chaque question, avec un code clair et **bien commenté**.
— Il faut l'appeler ainsi : Nom-Prénom-TP-Matrices. Le nom en majuscules, le prénom en minuscules.
— **L'unique** extension acceptable est : .py
— Les deux derniers exercices proviennent d'oraux de l'épreuve Maths-Info de Centrale-Supélec. Le jour J, vous n'aurez probablement pas accès à ChatGPT, alors mieux vaut essayer d'y répondre par soi-même.

Prérequis

Avant de commencer, assurez-vous d'avoir installé la bibliothèque `numpy`. Vous pouvez l'installer en utilisant la commande suivante dans votre terminal :

```
pip install numpy
```

Si c'est déjà fait, commencez par importer les bibliothèques nécessaires :

```
import numpy as np
import numpy.random as rd
```

Tableau des Opérations Matricielles en Python

Voici un tableau récapitulatif des principales opérations matricielles en Python avec `numpy` :

Opération	Fonction	Exemple en Python
Création d'une matrice	`np.array(Lignes de la matrices)`	`A = np.array([[1, 2], [3, 4]])` `print(A) # Affiche [[1 2] [3 4]]`

Addition de deux matrices A et B	A+B	`A = np.array([[1, 2], [3, 4]])`
		`B = np.array([[5, 6], [7, 8]])`
		`print(A + B) # Affiche [[6, 8], [10, 12]]`

Produit matriciel	`np.dot(A,B) ou A.dot(B)`	`A = np.array([[1, 2], [3, 4]])`
		`B = np.array([[5, 6], [7, 8]])`
		`print(np.dot(A, B)) # Affiche [[19, 22], [43, 50]]`

Transposition	`np.transpose(A)`	`A = np.array([[1, 2], [3, 4]])` `print(np.transpose(A)) # Affiche [[1, 3], [2, 4]]`
Déterminant	`np.linalg.det(A)`	`A = np.array([[1, 2], [3, 4]])` `print(np.linalg.det(A)) # Affiche -2.0`

Inverse	`np.linalg.inv(A)`	`A = np.array([[1, 2], [3, 4]])`
		`A_inv = np.linalg.inv(A)`
		`print(A_inv) # Affiche [[-2, 1], [1.5, -0.5]]`

Matrice identité de taille n	`np.eye(n)`	`I = np.eye(3) # Matrice identité 3x3`
Matrice nulle de taille $a \times b$	`np.zeros((a,b))`	`A = np.zeros((2,3)) # Matrice nulle 2x3` `print(A) # Affiche [[0, 0, 0], [0, 0, 0]]`
Diag$(a_0, \ldots, a_n)$	`np.diag([`$a_0, \ldots, a_n$`])`	`A = np.diag([1,2,3])` `print(A) # Affiche [[1, 0, 0], [0, 2, 0], [0, 0, 3]]`

Matrice contenant des valeurs aléatoires entre 0 et 1	`rd.random((nb de lignes, nb de colonnes))`	`A = rd.random((5,2))`
		
		`print(A) # Affiche une matrice aléatoire de 5 lignes et 2 colonnes dont les coefficients sont entre 0 et 1`

Donner une valeur n au coefficient (i, j) d'une matrice M	`M[i-1,j-1]=n (En python l'indexation des coefficients de la matrice commence à 0)`	`A = np.array([[2, 3], [1, 4]])`
		
		`A[0,0], A[1,1]=7, 5`
		`print(A) # Affiche [[7, 3], [1, 5]]`

Question 1 : Exécutez en python les exemples du tableau ci-dessus, et vérifiez que vous obtenez les mêmes résultats.

Question 2 : Soient `A = np.array([[2, 3], [1, 4]])` et `B = np.array([[1, 5], [2, 7]])` Comparer le résultat de $A * B$ et `np.dot(A,B)`.

Question 3 : Considérons `M = np.array([[2, 3, 0], [1, 4, 5], [7, 8, 9]])`. Affichez le résultat de np.shape(M) et np.size(M). En déduire le rôle de chaque fonction.

Question 4 : Générez une matrice aléatoire de taille 5×6 dont les valeurs sont comprises entre 0 et 100.

Question 5 : Affichez la matrice suivante

$$\begin{pmatrix} 0 & 1.0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0 & 0.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3 & 0 & 0.7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.4 & 0 & 0.6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0.4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.8 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.9 & 0 & 0.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.0 & 0 \end{pmatrix}$$

Exercice 1 : Les matrices et les suites

On considère trois suites réelles (x_n), (y_n), (z_n) vérifiant

$$x_0 = y_0 = \frac{1}{2} \text{ et } z_0 = 0.$$

On suppose que

$$\begin{cases} 10x_{n+1} = 7x_n + 4y_n + 5z_n \\ 10y_{n+1} = 3x_n + z_n \\ 10z_{n+1} = 6y_n + 4z_n \end{cases}$$

À l'aide d'un programme en Python, calculer $(x_{2018}, y_{2018}, z_{2018})$ par un calcul matriciel.

Exercice 2 : Matrice compagnon

Soient d un entier naturel ≥ 2, $a_0, \ldots, a_{d-1}$ des entiers, posons $P = X^d + a_{d-1}X^{d-1} + \cdots + a_1X + a_0$. On associe à P la matrice :

$$C_P = \begin{pmatrix} 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & \ddots & \vdots & -a_2 \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & 1 & -a_{d-1} \end{pmatrix}$$

C_p s'appelle la matrice compagnon associée au polynôme P.

Écrire une fonction qui retourne la matrice compagnon associée à un polynôme P, et afficher le résultat pour $P = X^4 + 5X^3 - X + 7$.

Exercice 3 : Résolution de Systèmes Linéaires

Soient n un entier supérieur ou égal à 2, $A \in \mathfrak{M}_n(\mathbb{R})$ et $B \in \mathfrak{M}_{n,1}(\mathbb{R})$. On cherche à résoudre numériquement le système linéaire $AX = B$ selon la méthode de Jacobi. On fait l'hypothèse que les coefficients diagonaux de A sont non nuls. On pose

$$A = D - E - F,$$

où D est diagonale, E triangulaire strictement supérieure et F triangulaire strictement inférieure. On se donne $X_0 \in \mathfrak{M}_{n,1}(\mathbb{R})$ et on définit la suite $(X_k)_{k\geq 0}$ par la récurrence :

$$\forall k \in \mathbb{N}, \quad X_{k+1} = D^{-1}(E+F)X_k + D^{-1}B.$$

(a) Écrire la fonction decomp(A) qui renvoie D^{-1}, E et F.

(b) Écrire la fonction Jacobi(A, B, X_0, n) qui renvoie X_n lorsque $n \in \mathbb{N}$.

(c) Tester la fonction lorsque

$$A = \begin{pmatrix} 2 & 3 & 1 \\ -2 & 5 & 4 \\ 0 & 1 & 8 \end{pmatrix}, \quad B = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad \text{et} \quad X_0 = B.$$

Comparer le résultat obtenu avec celui obtenu à l'aide de numpy.linalg.solve(A, B) (fonction prédéfinie sur python pour résoudre les systèmes linéaires).