**Chapitre 4 : Algèbre linéaire – Tableaux**

Ce chapitre m’a permis de mieux comprendre l’importance de l’algèbre linéaire dans le calcul scientifique. Ce n’est pas juste une matière abstraite : en pratique, les **vecteurs** et les **matrices** sont partout. Avec **NumPy**, Python nous donne les outils pour manipuler ces objets de façon simple et rapide.

**1. Les bases des tableaux**

On commence par apprendre à créer des vecteurs et des matrices à l’aide de np.array(). Par exemple :

python

CopierModifier

import numpy as np

A = np.array([[1, 2], [3, 4]])

C’est comme ça qu’on définit une matrice 2x2. NumPy permet aussi de créer des tableaux vides, remplis de zéros ou d’uns, ou même d’utiliser arange() ou linspace() pour créer des séquences numériques.

Ce que j’ai aimé, c’est qu’on peut accéder facilement aux éléments avec l’indexation classique (A[0][1], etc.) mais aussi avec le **slicing** pour extraire des sous-matrices.

**2. Les opérations mathématiques**

Les opérations sont vectorisées : pas besoin de boucles. On peut faire :

* l’addition ou la soustraction de matrices
* la multiplication élément par élément : A \* B
* ou le produit matriciel classique : A @ B ou np.dot(A, B)

C’est là que j’ai vraiment compris que le @ était pour le produit matriciel, pas une multiplication simple.

**3. Transposition, inversion, déterminant**

J’ai découvert des opérations classiques :

* Transposer une matrice : A.T
* Calculer l’inverse d’une matrice : np.linalg.inv(A)
* Obtenir le déterminant : np.linalg.det(A)

C’est la partie où l’on utilise le module numpy.linalg, qui est spécialisé pour tout ce qui est **algèbre linéaire**.

**4. Résolution de systèmes linéaires**

C’est une des applications pratiques les plus utiles : résoudre une équation du type Ax = b.

Plutôt que d’inverser A manuellement (ce qui est instable numériquement), on utilise :

python

CopierModifier

x = np.linalg.solve(A, b)

Ça permet de trouver le vecteur x qui satisfait l’équation, directement.

**5. Valeurs propres et vecteurs propres**

J’ai aussi vu qu’on pouvait calculer les **valeurs propres** (eigenvalues) et les **vecteurs propres** (eigenvectors) avec :

python

CopierModifier

valeurs, vecteurs = np.linalg.eig(A)

Ce sont des concepts que j’ai vus en cours, mais c’est la première fois que je les applique en code.

**Ce que je retiens**

* NumPy est parfaitement adapté à l’algèbre linéaire.
* Il faut distinguer multiplication élément par élément (\*) et produit matriciel (@).
* Le module linalg est la boîte à outils pour toutes les opérations avancées (résolution, inversion, valeurs propres, etc.).
* Tous ces outils sont essentiels dans des domaines comme l’analyse de données, les statistiques, le machine learning, etc.