Université de AbdelHafid Boussouf - Mila

# CHAPITRE N° 5:

# Le calcul scientifique en Python avec Le module **NumPy**

**L1 Mathématiques appliquées**

# Présentation des bibliothèques

- Python ne sait pas faire grand chose dans le domaine mathématique, **comme tracer une fonction, calculer des valeurs de fonctions usuelles, réaliser des opérations matricielles.**

- Cependant, de nombreux modules ont été développés pour pallier ce manque, parmi lesquels :
  - Module numpy
  - Module matplotlib

# Introduction

Python ne propose de base que le type **list**, conteneur dynamique hétérogène puissant, mais elle est plus difficile dans le contexte des mathématiques (pas orienté calcul numérique).

Par exemple l'opérateur "*" ou "+" ne correspond pas à la multiplication /somme d'un vecteur avec un nombre.

Le module numpy a été créé pour résoudre ce problème. En plus d'un grand nombre de fonctions, elle définit un autre type de tableaux, spécialement conçus pour les opérations mathématiques, tels que vecteurs, matrices.

# Le module numpy

- La bibliothèque "numpy" permet de manipuler facilement les tableaux (array); vecteurs ou matrices.

- Les tableaux (array) "numpy" ne gère que les objets de même type.

- La bibliothèque "numpy" propose un grand nombre de fonctions comme :
  - L'accès rapide aux données d'une matrice ou d'un vecteur,
  - La recherche, et l'extraction,
  - Le tri de données,
  - Calcul statistique de données (moyenne, médiane, somme,...),
  - Calcul scientifique (algèbre linéaire).

# Les sous-modules numpy

Le module numpy propose un ensemble de classes, d'objets et de fonctions dédiés aux calculs numériques :

- Classe ndarray : Créer des tableaux homogène multi-dimensionnels.
- Classe matrix : Permet de faire de calcul matriciel.
- numpy.linalg : Un module d'algèbre linéaire basique.
- numpy.random : Un module pour les générateurs aléatoires.

# Importer numpy

Afin d'utiliser le module "numpy" il faut tout d'abord l'importer via l'instruction suivante :

```
import numpy as np
```

NB : Si le nom du module est un peu long il est préférable de créer un alias via l'instruction as dans notre cas c'est np.

# Manipulation du module numpy : Création ndarray

## Création d'objets ndarray :

```
]: import numpy as np

# Création des matrices
m1 = np.ndarray((2,2))                    # float par défaut
m2 = np.ndarray((2,2), dtype=int)         # entier
m3 = np.ndarray((2,2), dtype=bool)        # booléen

# Affichage
print("m1 =\n", m1)
print("m2 =\n", m2)
print("m3 =\n", m3)
```

```
m1 =
 [[2.12199579e-314 6.95314361e-310]
 [4.42682819e-321 6.91770628e-312]]
m2 =
 [[      4294967296 140733193388032]
 [             896    140159339264]]
m3 =
 [[ True  True]
 [ True  True]]
```

La fonction np.ndarray() crée une matrice mais elle contient des valeurs non initialisées (aléatoires).



**Remarque**: Les valeurs peuvent changer à chaque exécution ⚠

# Manipulation du module numpy : Création d'objets matrix

La classe np.matrix  permet d'appliquer des méthodes spécifiques aux matrices.

```python
import numpy as np

# Création d'une matrice avec np.matrix
m = np.matrix([[1,2,3],[4,5,6],[7,8,9]])

# Affichage de la matrice
print("La matrice m :")
print(m)

# Afficher la dimension de la matrice
print("\nDimensions :", m.shape)

# Accès à un élément (ligne 1, colonne 2)
print("\nÉlément (1,2) :", m[1,2])

# Exemple d'opération : multiplication matricielle
print("\nMultiplication m * m :")
print(m * m)

# Transposée de la matrice
print("\nTransposée :")
print(m.T)
```

Exécution:

```
La matrice m :
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Dimensions : (3, 3)

Élément (1,2) : 6

Multiplication m * m :
[[ 30  36  42]
 [ 66  81  96]
 [102 126 150]]

Transposée :
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

# Manipulation du module numpy : Fonction array()

La fonction array() convertit un objet list en objet ndarray.

```
import numpy as np

# 1. Conversion d'une liste en array
v1 = np.array([1,3,5,6,7])

# 2. Conversion avec type float
v2 = np.array([1,3,5,6,7], float)

# 3. Tableau à deux dimensions (matrice)
m = np.array([[1,3,5,6,7],[3,8,9,1,0]], float)

# Affichage
print("v1 =")
print(v1)

print("\nv2 =")
print(v2)

print("\nm =")
print(m)
```

Exécution:

```
v1 =
[1 3 5 6 7]

v2 =
[1. 3. 5. 6. 7.]

m =
[[1. 3. 5. 6. 7.]
 [3. 8. 9. 1. 0.]]
```

# Manipulation du module numpy : Fonctions arange() et linspace()

Les fonctions **arange**(début, fin, pas) et **linspace**(début, fin, nbElement): permettent la création d'un vecteur correspondant à progression arithmétique.

```
|:  import numpy as np

    # Utilisation de arange
    v1 = np.arange(1, 10, 2)

    # Utilisation de linspace
    v2 = np.linspace(1, 10, 5)

    # Affichage
    print("arange(1,10,2) =")
    print(v1)

    print("\nlinspace(1,10,5) =")
    print(v2)
```

Exécution:

```
arange(1,10,2) =
[1 3 5 7 9]

linspace(1,10,5) =
[ 1.    3.25  5.5   7.75 10.  ]
```

# Manipulation du module numpy : La fonction reshape()

La fonction reshape(n,p) convertit un vecteur en une matrice.

```python
import numpy as np

# Création d'un vecteur avec arange
v = np.arange(1, 7)

print("Vecteur v :")
print(v)

# Transformation en matrice 2 lignes × 3 colonnes
m = v.reshape(2, 3)

print("\nMatrice m après reshape(2,3) :")
print(m)
```

Exécution:

```
Vecteur v :
[1 2 3 4 5 6]

Matrice m après reshape(2,3) :
[[1 2 3]
 [4 5 6]]
```

# Manipulation du module numpy : Les vecteurs particuliers

Les fonctions zeros(n), ones(n) et rand(n) permettent la création des vecteurs nulles, vecteurs remplies de 1, et vecteurs remplie à coefficients aléatoires.

```python
import numpy as np

v = np.array([]) # Création d'un vecteur vide
print("Vecteur vide :",v)

# Vecteur de zéros
v1 = np.zeros(5)
print("\nVecteur de zéros :",v1)

# Vecteur de uns (1)
v2 = np.ones(5)
print("\nVecteur de uns :",v2)

# Vecteur aléatoire
v3 = np.random.rand(5)
print("\nVecteur aléatoire :",v3)
```

Exécution:

```
Vecteur vide : []

Vecteur de zéros : [0. 0. 0. 0. 0.]

Vecteur de uns : [1. 1. 1. 1. 1.]

Vecteur aléatoire : [0.62354971 0.22625258
0.98902679 0.18997713 0.28545598]
```

# Manipulation du module numpy : Les matrices particulières

Les fonctions zeros((n,p)), ones((n,p)), eye(n), diag(v) et rand((n,p)) permettent la création des matrices nulles, remplies de 1, d'identités, dont le diagonale un vecteur et matrice remplie à coefficients aléatoires.

```
import numpy as np
m1 = np.zeros((2,3))  # Matrice de zéros
m2 = np.ones((2,3))   # Matrice de uns (1)
m3 = np.eye(3)        # Matrice identité
v = [1,2,3] # Matrice diagonale à partir d'un vecteur
m4 = np.diag(v)
# Matrice aléatoire
m5 = np.random.rand(2,3)
# Affichage
print("Matrice de zéros :")
print(m1)
print("\nMatrice de uns :")
print(m2)
print("\nMatrice identité :")
print(m3)
print("\nMatrice diagonale :")
print(m4)
print("\nMatrice aléatoire :")
print(m5)
```

Exécution:

```
Matrice de zéros :
[[0. 0. 0.]
 [0. 0. 0.]]

Matrice de uns :
[[1. 1. 1.]
 [1. 1. 1.]]

Matrice identité :
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

Matrice diagonale :
[[1 0 0]
 [0 2 0]
 [0 0 3]]

Matrice aléatoire :
[[0.28604998 0.08269743 0.76400208]
 [0.06431091 0.40962595 0.67101565]]
```

# Manipulation du module numpy: accès vecteur

On accède aux éléments d'un vecteur par indice. Les indices commencent à 0 et finissent à N-1.

On accède à un élément du vecteur avec la notation : Vec[indice]

```
import numpy as np
v = np.array([1,2,7,9,3,5,6]) # Création du vecteur
print("Vecteur v :",v)
print("Premier élément v[0] :",v[0]) # Récupérer le premier élément
print("Dernier élément v[-1] :",v[-1]) # Récupérer le dernier élément
print("Sous-vecteur v[1:5] :",v[1:5])# Récupérer une partie du vecteur (indices 1 à 4)
print("Tous les éléments v[:] :",v[:]) # Récupérer tous les éléments
```

```
Vecteur v : [1 2 7 9 3 5 6]
Premier élément v[0] : 1
Dernier élément v[-1] : 6
Sous-vecteur v[1:5] : [2 7 9 3]
Tous les éléments v[:] : [1 2 7 9 3 5 6]
```

# Manipulation du module numpy: L'accès matrice

On accède aux éléments d'une matrice par indice i de ligne et indice de colonne j.
Notation : Mat[i,j]

```python
import numpy as np

# Création de la matrice 3x3
m1 = np.array([[1,2,3],
               [4,5,6],
               [7,8,9]])
print("Matrice m1 :",m1)

# Récupérer un élément (ligne 1, colonne 2)
print("\nélément m1[1,2] :",m1[1,2])

# Récupérer toute une ligne
print("\nLigne 2 m1[1] :",m1[1])

# Récupérer la dernière ligne
print("\nDernière ligne m1[-1] :",m1[-1])

# Récupérer une ligne spécifique
print("\nLigne 3 m1[2,:] :",m1[2,:])

# Récupérer une colonne
print("\nColonne 2 m1[:,1] :",m1[:,1])

# Récupérer une sous-matrice
print("\nSous-matrice m1[1:3, 0:3] :",m1[1:3, 0:3])
```

Exécution:

```
Matrice m1 : [[1 2 3]
 [4 5 6]
 [7 8 9]]

Élément m1[1,2] : 6

Ligne 2 m1[1] : [4 5 6]

Dernière ligne m1[-1] : [7 8 9]

Ligne 3 m1[2,:] : [7 8 9]

Colonne 2 m1[:,1] : [2 5 8]

Sous-matrice m1[1:3, 0:3] : [[4 5 6]
 [7 8 9]]
```

# Les opérations sur les tableaux numpy

Les opérations usuelles (addition, soustraction, multiplication, division) s'appliquent aux tableaux Numpy (Vecteur/Matrice) en opérant coefficient par coefficient.

```python
import numpy as np

# Création des matrices
a = np.array([[1,4],[2,3]])
b = np.ones((2,2))
print("Matrice a :")
print(a)
print("\nMatrice b :")
print(b)
# Multiplication (élément par élément)
print("\nProduit a * b :")
print(a * b)
# Addition
print("\nAddition a + b :")
print(a + b)
# Soustraction
print("\nSoustraction a - b :")
print(a - b)
# Division
print("\nDivision a / b :")
print(a / b)
# Puissance
print("\nPuissance a**2 :")
print(a ** 2)
```

Exécution:

```
Matrice a :
[[1 4]
 [2 3]]

Matrice b :
[[1. 1.]
 [1. 1.]]

Produit a * b :
[[1. 4.]
 [2. 3.]]

Addition a + b :
[[2. 5.]
 [3. 4.]]

Soustraction a - b :
[[0. 3.]
 [1. 2.]]

Division a / b :
[[1. 4.]
 [2. 3.]]

Puissance a**2 :
[[ 1 16]
 [ 4  9]]
```

# Les méthodes sur les tableaux numpy (01)

| Méthode | Description |
|---|---|
| np.size(V/M) | Renvoie le nombre d'éléments d'un vecteur ou d'une matrice. |
| np.shape(M) | Renvoie les dimensions d'une matrice |
| np.sum(V/M) | Somme des éléments d'un vecteur ou d'une matrice |
| np.sum(M,axis=0) | Somme des lignes d'une matrice (addition vertical). |
| np.sum(M,axis=1) | Somme des colonne d'une matrice (addition horizontal). |
| np.vdot(V1,V2) | Produit scalaire de deux vecteurs |
| V.min(); V.max() ; V.sum() | Minimum, maximum et somme des éléments d'un vecteur |
| M.min();M.max() ; M.sum() | Minimum, maximum et somme des éléments d'une matrice |
| M.sum(axis=0);M.min(axis=0) | Somme des lignes et minimum des lignes d'une matrice |
| M.cumsum(axis=0); M.cumsum(axis=1) | somme cumulée des lignes et somme cumulée des colonnes |

# Les méthodes sur les tableaux numpy(02)

| Méthode | Description |
|---|---|
| np.cross(V1,V2) | Produit vectoriel de deux vecteurs. |
| np.linalg.inv(M) | Inverse d'une matrice |
| np.transpose(M) | Transposée d'une matrice |
| np.dot(M1,M2) | Produit matriciel |
| np.linalg.det(M) | Déterminant d'une matrice |
| np.diag() | Création Matrice diagonale |
| np.trace(M) | La trace d'une matrice |
| np.linalg.matrix_power(M,n) | M à la puissance n |
| np.sin(M); np.cos(M); np.tan(M) | Les fonctions trigonométriques de base |
| np.linalg.solve(A,B) | résolution du système linéaire A x = B |

# Conclusion

Dans ce chapitre, nous avons découvert la bibliothèque NumPy, qui est très importante en Python pour le calcul scientifique.

☞ Nous avons appris que :

- ❑NumPy permet de travailler avec des tableaux (array).
- ❑Il facilite les calculs sur les vecteurs et les matrices.
- ❑Les opérations sont simples et rapides (+ , - , * , /).
- ❑Il offre plusieurs fonctions utiles :
  - ▪array() pour créer des tableaux.
  - ▪arange() et linspace() pour créer des valeurs.
  - ▪reshape() pour transformer les données.
  - ▪zeros(), ones() pour créer des tableaux particuliers.