DATA SCIENCE AVEC PYTHON

Année académique 2024-2025

Master: X-MAS-DE-4

Enseignant : David Rhenals

Vecteurs et matrices multidimensionnelles avec NumPy

Contenu

- Qu'est-ce que Numpy?
- A quoi sert Numpy?
- Les avantages à utiliser Numpy
- Attributs de base sur l'objet ndarray (NumPy)
- Comment créer des vecteurs et des matrices sur NumPy
- Qu'est ce que l'indexation et le découpage, et comment s'en servir
- Récapitulatif des principales opérations matricielles sur NumPy
- Quelques exemples introductifs à NumPy
  - Création de vecteurs et de matrices
    - Création de vecteurs et de matrices à partir de listes
    - Création de vecteurs et de matrices avec valeurs constantes
  - Indexation et découpage (Indexing and slicing)
  - Opérations Matricielles
- Mise en pratique de NumPy à travers des exercices simples sur Jupyter Notebooks

Qu'est-ce que Numpy

- **NumPy** est une librairie ou package python écrit en langage C qui permet la manipulation et la gestion d'opérations sur des vecteurs et des matrices multidimensionnelles

- La bibliothèque **Numpy** introduit un objet de classe clé nommé **ndarray** qui est optimisé pour les calculs numériques et offre une performance supérieure aux listes classiques du langage python

A quoi sert Numpy?

- **Manipulation de tableaux multidimensionnels**:
- **Opérations mathématiques:** NumPy fournit un vaste ensemble de fonctions mathématiques optimisées pour les opérations sur les tableaux. Cela inclut des opérations arithmétiques de base (addition, soustraction, multiplication, division), des fonctions trigonométriques, des fonctions exponentielles, des statistiques, etc.
- **Algèbre linéaire:** NumPy intègre des outils pour effectuer des opérations d'algèbre linéaire comme le produit matriciel, la résolution de systèmes d'équations linéaires, le calcul de valeurs propres et de vecteurs propres
- **Génération de nombres aléatoires:** NumPy offre des fonctions pour générer des nombres aléatoires selon différentes distributions (normale, uniforme, etc.)
- **Intégration avec d'autres bibliothèques:** NumPy est la base de nombreuses autres bibliothèques scientifiques en Python comme SciPy, Pandas, Matplotlib, et est donc largement utilisée dans les domaines de la data science, du machine learning et de l'analyse de données

Avantages à utiliser NumPy

- **Performance** : Les opérations sur les tableaux NumPy sont généralement beaucoup plus rapides que les opérations équivalentes sur les listes Python.
- **Concision** : NumPy permet d'écrire du code plus concis et plus lisible pour les opérations mathématiques.
- **Écosystème riche** : NumPy s'intègre parfaitement avec d'autres bibliothèques Python, offrant un écosystème complet pour l'analyse de données.

Attributs de base sur l'objet ndarray (NumPy)

- L'objet de classe **ndarray** est caractérisé pour les attributs basiques ci-dessous

Attributs basiques de l'objet ndarray NumPy

Attribut	Description
shape	Une tuple contenant le nombre d'éléments (ie. La longueur) pour chaque dimension de l'objet
size	Nombre total des éléments de l'objet
ndim	Nombre de dimensions de l'objet
nbytes	Nombre de bytes utilisés pour stocker les données
dtype	Le type de données des éléments stockés dans l'objet

Types de données numériques disponibles en NumPy

dtype	variants	Description
int	Int8, int16, int32, int64	Entiers
uint	uint8, uint16, uint32, uint64	Entiers non-negatifs
bool	Bool	Bololean (Vrai ou Faux)
float	float16, float32, float64, float128	Nombres à virgule flottante
complex	complex64, complex128, complex256	Nombres complexe à virgule flottante

Comment créer des vecteurs et des matrices avec NumPy

Plus de fonctionnalités sur la création d'objets ndarrays

Fonction	Description
np.array	Crée un tableau dont les éléments sont donnés par un objet de type tableau, qui, par exemple, peut être une liste Python (imbriquée), un tuple, une séquence itérable ou une autre instance de ndarray.
np.zeros	Crée un tableau avec les dimensions et le type de données spécifiés qui est rempli avec des zéros.
np.ones	Crée un tableau avec les dimensions et le type de données spécifiés, qui est rempli avec le nombre 1.
np.diag	Crée un tableau diagonal avec des valeurs spécifiées le long de la diagonale et des zéros ailleurs.
np.arange	Crée un tableau avec des valeurs régulièrement espacées entre les valeurs de début, de fin et d'incrémentation spécifiées.
np.linspace	Crée un tableau avec des valeurs régulièrement espacées entre les valeurs de début et de fin spécifiées, en utilisant un nombre spécifié d'éléments.
np.logspace	Crée un tableau dont les valeurs sont logarithmiquement espacées entre les valeurs de départ et d'arrivée.
np.meshgrid	Génère des matrices de coordonnées (et des tableaux de coordonnées de dimension supérieure) à partir de vecteurs de coordonnées unidimensionnels.
np.fromfunction	Crée un tableau et le remplit avec les valeurs spécifiées par une fonction donnée, qui est évaluée pour chaque combinaison d'indices pour la taille de tableau donnée.
np.fromfile	Crée un tableau avec les données d'un fichier binaire (ou texte).
np.genfromtxt, np.loadtxt	Créez un tableau à partir de données lues dans un fichier texte, par exemple un fichier CSV (comma separated value). La fonction np.genfromtxt prend également en charge les fichiers de données avec des valeurs manquantes.
np.random.rand	Génère un tableau de nombres aléatoires uniformément distribués entre 0 et 1. D'autres types de distributions sont également disponibles dans le module np.random.

Qu'est ce que l'indexation et le découpage, et comment s'en servir

- **L'Indexation** est un mécanisme permettant d'accèder à des éléments individuels d'un **ndarray** en utilisant leur position numérique.
- **Le Découpage (Slicing)** est une opération plus puissante permettant d'extraire de sous-séquences (des tranches) d'un **ndarray** en spécifiant un intervalle d'indices.

Expression	Description
b[m]	Sélectionne l'élément à l'index m, où m est un nombre entier (commence à compter à partir de 0).
b[-m]	Sélectionner le n ième élément de la fin de la liste, où n est un nombre entier. Le dernier élément de la liste est désigné par -1, l'avant-dernier élément par -2, et ainsi de suite.
b[m:n]	Sélectionne les éléments dont l'indice commence à m et se termine à n - 1 (m et n sont des nombres entiers).
b[:]	Sélectionne tous les éléments de l'axe donné (les axes font référence aux lignes et aux colonnes) .
b[:n]	Sélectionne les éléments commençant par l'indice 0 et allant jusqu'à l'indice n -1 (entier).
b[m:]	Sélectionne les éléments à partir de l'indice m (entier) et jusqu'au dernier élément du tableau.
b[m:n:p]	Sélectionne les éléments d'indice m à n (exclusif), avec l'incrément p.
b[::-1]	Sélectionne tous les éléments, dans l'ordre inverse.

Récapitulatif des principales opérations matricielles sur NumPy

Opérations élémentaires

Opération mathématique	Notation mathématique	Notation NumPy	Description
Addition de matrices	$A + B$	**np.add(A, B) ou A + B**	Somme élément par élément de deux matrices
Soustraction de matrices	$A - B$	**np.subtract(A, B) ou A - B**	Différence élément par élément de deux matrices
Multiplication par un scalaire	$k * A$	**k * A**	Multiplication de chaque élément de la matrice par un scalaire
Multiplication matricielle	$A * B$	**np.dot(A, B) ou A @ B**	Produit matriciel de deux matrices (A*B!=B*A)

Plus de fonctionnalités np.linalg module

Quelques opérations d'algèbre linéaire

Opération Mathématique	Notation mathématique	Notation Numpy/Scipy	Description
Transposée	A^T	**A.T**	Matrice obtenue en échangeant les lignes et les colonnes de A.
Déterminant	$\det(A)$	**np.linalg.det(A)**	Scalaire associé à une matrice carrée A, calculé à partir des combinaisons linéaires des éléments de A.
Inverse	A^{-1}	**np.linalg.inv(A)**	Matrice B telle que A * B = B * A = I (matrice identité), si A est inversible.
Rang	$rang(A)$	**np.linalg.matrix_rank(A)**	Nombre maximal de lignes (ou de colonnes) linéairement indépendantes de A.
Trace	$tr(A)$	**np.trace(A)**	Somme des éléments de la diagonale principale de la matrice carrée A.
Résolution de AX = B	$X = A^{-1}B$	**np.linalg.solve(A, B)**	Résolution du système d'équations linéaires AX = B, où A est une matrice carrée inversible, X le vecteur des inconnues et B le vecteur des termes constants.
Valeurs et vecteurs propres	λ, v	**np.linalg.eig(A)**	Scalaires λ (valeurs propres) et vecteurs non nuls v (vecteurs propres) tels que Av = λv.
Décomposition en valeurs singulières (SVD)	$A = USV^H$	**np.linalg.svd(A)**	Décomposition d'une matrice A en le produit de trois matrices : U (matrice orthogonale), S (matrice diagonale des valeurs singulières) et V^H (conjuguée de la transposée de V, matrice orthogonale).
Racine carrée élément par élément	$\sqrt{A}$	**np.sqrt(A)**	Calcule la racine carrée de chaque élément de A.
Exponentielle élément par élément	e^A	**np.exp(A)**	Calcule l'exponentielle de chaque élément de A.
Logarithme naturel élément par élément	$\log(A)$	**np.log(A)**	Calcule le logarithme naturel de chaque élément de A.

Quelques exemples introductifs à NumPy (Création de vecteurs et de matrices à partir de listes)

Vecteur 1D

```
[2]: # Import de la bibliothèque NumPy avec alias np
     import numpy as np
     # Création d'un vecteur (1d) à partir d'une liste
     vecteur = np.array([1, 2, 3, 4, 5])
     # Création et affichage du dictionnaire avec les attributs ndarray (vecteur)
     dict(
         vecteur=vecteur,
         shape=vecteur.shape,
         size=vecteur.size,
         dim=vecteur.ndim,
         nbytes=vecteur.nbytes,
         dtype=vecteur.dtype
     )

[2]: {'vecteur': array([1, 2, 3, 4, 5]),
      'shape': (5,),
      'size': 5,
      'dim': 1,
      'nbytes': 20,
      'dtype': dtype('int32')}
```

Liste simple [,]

Matrice 2D

```
[3]: # Création d'une matrice (2d) à partir d'une liste
     matrice2d = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
     # Création et affichage du dictionnaire avec les attributs ndarray (matrice 2d)
     dict(
         matrice2d=matrice2d,
         shape=matrice2d.shape,
         size=matrice2d.size,
         dim=matrice2d.ndim,
         nbytes=matrice2d.nbytes,
         dtype=matrice2d.dtype,
     )

[3]: {'matrice2d': array([[ 1,  2,  3,  4,  5],
            [ 6,  7,  8,  9, 10]]),
      'shape': (2, 5),
      'size': 10,
      'dim': 2,
      'nbytes': 40,
      'dtype': dtype('int32')}
```

Liste imbriqué [[,],[,]]

Liste doublement imbriqué [[[,],[,]],[[,],[,]]]

Matrice 3D

```
[4]: # Création d'une matrice (2d) à partir d'une liste
     matrice3d = np.array(
         [
             [[1, 5, 2, 1], [4, 3, 5, 6], [6, 3, 0, 6], [7, 3, 5, 0], [2, 3, 3, 5]],
             [[2, 2, 3, 1], [4, 0, 0, 5], [6, 3, 2, 1], [5, 1, 0, 0], [0, 1, 9, 1]],
             [[3, 1, 4, 2], [4, 1, 6, 0], [1, 2, 0, 6], [8, 3, 4, 0], [2, 0, 2, 8]],
         ]
     )
     # Création et affichage du dictionnaire avec les attributs ndarray (matrice 2d)
     dict(
         matrice3d=matrice3d,
         shape=matrice3d.shape,
         size=matrice3d.size,
         dim=matrice3d.ndim,
         nbytes=matrice3d.nbytes,
         dtype=matrice3d.dtype,
     )

[4]: {'matrice3d': array([[[1, 5, 2, 1],
             [4, 3, 5, 6],
             [6, 3, 0, 6],
             [7, 3, 5, 0],
             [2, 3, 3, 5]],

            [[2, 2, 3, 1],
             [4, 0, 0, 5],
             [6, 3, 2, 1],
             [5, 1, 0, 0],
             [0, 1, 9, 1]],

            [[3, 1, 4, 2],
             [4, 1, 6, 0],
             [1, 2, 0, 6],
             [8, 3, 4, 0],
             [2, 0, 2, 8]]]),
      'shape': (3, 5, 4),
      'size': 60,
      'dim': 3,
      'nbytes': 240,
      'dtype': dtype('int32')}
```

Quelques exemples introductifs à NumPy (Création de vecteurs et de matrices avec valeurs constantes)

Dimension k

Dimension i (lignes)

Dimension j (colonnes)

Éléments de la matrice définis comme complexe

```
[10]: # Création d'une matrice 3D de zéros avec type de données complèxes
      np.zeros(shape=(3, 3, 2), dtype=complex)

[10]: array([[[0.+0.j, 0.+0.j],
              [0.+0.j, 0.+0.j],
              [0.+0.j, 0.+0.j]],

             [[0.+0.j, 0.+0.j],
              [0.+0.j, 0.+0.j],
              [0.+0.j, 0.+0.j]],

             [[0.+0.j, 0.+0.j],
              [0.+0.j, 0.+0.j],
              [0.+0.j, 0.+0.j]]])
```

Liste d'éléments complexes conformant la diagonale principale

Diagonale en question si k=0 diagonale principale, si k<0 diagonale en dessous de la diagonale principale, si k>0 diagonale en dessus de la diagonale principale

```
[11]: # Création d'un matrice 2D diagonale complèxe
      np.diag(v=[1.0 + 4j, 4 + 8j, 6 + 1j, 0.1 + 0.3j], k=0)

[11]: array([[1. +4.j , 0. +0.j , 0. +0.j , 0. +0.j ],
             [0. +0.j , 4. +8.j , 0. +0.j , 0. +0.j ],
             [0. +0.j , 0. +0.j , 6. +1.j , 0. +0.j ],
             [0. +0.j , 0. +0.j , 0. +0.j , 0.1+0.3j]])
```

Fonction travaillant uniquement sur des matrices carrées (m = n)

Éléments de la matrice définis comme des nombres à virgule flottant

```
[6]: # Création de matrice 2D complèxe.
     # Cette fonction travaille avec des matrices carrées
     np.identity(n=5, dtype=float)

[6]: array([[1., 0., 0., 0., 0.],
            [0., 1., 0., 0., 0.],
            [0., 0., 1., 0., 0.],
            [0., 0., 0., 1., 0.],
            [0., 0., 0., 0., 1.]])
```

Liste d'éléments complexes conformant la diagonale en question

Diagonale principale décalée d'une unité

```
[8]: # Création d'un matrice 2D diagonale complèxe
     np.diag(v=[1.0 + 4j, 4 + 8j, 6 + 1j, 0.1 + 0.3j], k=1)

[8]: array([[0. +0.j , 1. +4.j , 0. +0.j , 0. +0.j , 0. +0.j ],
            [0. +0.j , 0. +0.j , 4. +8.j , 0. +0.j , 0. +0.j ],
            [0. +0.j , 0. +0.j , 0. +0.j , 6. +1.j , 0. +0.j ],
            [0. +0.j , 0. +0.j , 0. +0.j , 0. +0.j , 0.1+0.3j],
            [0. +0.j , 0. +0.j , 0. +0.j , 0. +0.j , 0. +0.j ]])
```

Quelques exemples introductifs à NumPy (Indexation et découpage)

Indexation et découpage appliqués à un vecteur 1D

```
[7]: # Création d'un vecteur 10 éléments
     # (la fonction crée le vecteur m=0 jusqu'a n-1)
     a = np.arange(0, 11)
     # Dictionnaire affichant les éléments définis pour le dictionnaire
     dict(
         a=a,  # Vecteur complet
         premier_element=a[0],  # Séléction du premier élément
         dernier_element=a[-1],  # Séléction du dernier élément
         cinquieme_element=a[4],  # Séléction du cinquieme élément
         tous_elements=a[:],   # Séléction de tous les éléments du vecteur
         deuxieme_avant_denier_element=a[1:-1],  # Séléction du deuxième à l'avant dernier élément
         trois_denier_element=a[-3:],  # Séléction des trois derniers éléments
         cinq_premier_elemets=a[:5],  # Séléction des cinqs premiers éléments
         premier_dernier_pas_2=a[0::2]  # Séléction des éléments à un pas de deux en incluant le dernier élément
     )

[7]: {'a': array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
      'premier_element': 0,
      'dernier_element': 10,
      'cinquieme_element': 4,
      'tous_elements': array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]),
      'deuxieme_avant_denier_element': array([1, 2, 3, 4, 5, 6, 7, 8, 9]),
      'trois_denier_element': array([ 8,  9, 10]),
      'cinq_premier_elemets': array([0, 1, 2, 3, 4]),
      'premier_dernier_pas_2': array([ 0,  2,  4,  6,  8, 10])}
```

Indexation et découpage appliqués à une matrice 2D

```
[8]: # Création d'une fonction lambda
     f = lambda m, n : m + 2*n
     # Création d'une matrice 6*6 à partir de la fonction lambda ci-dessus
     B = np.fromfunction(f, [6, 6], dtype=int)
     # Création d'un dictionnaire pour affichage des résultats
     dict(
         B=B,  # Matrice complète
         premiere_colonne=B[:, 0],  # Sélection de la première colonne
         derniere_colonne=B[:, -1],  # Sélection de la dernière colonne
         premier_ligne=B[0, :],  # Sélection de la première ligne
         toute_matrice=B[:,:],  # Sélection de tous les éléments de la matrice
         colonne_deux_avant_derniere=B[:,1:-1]  # Sélection depuis le deuxième colonne jusqu'à l'avant dernière
     )

[8]: {'B': array([[ 0,  2,  4,  6,  8, 10],
                  [ 1,  3,  5,  7,  9, 11],
                  [ 2,  4,  6,  8, 10, 12],
                  [ 3,  5,  7,  9, 11, 13],
                  [ 4,  6,  8, 10, 12, 14],
                  [ 5,  7,  9, 11, 13, 15]]),
       'premiere_colonne': array([0, 1, 2, 3, 4, 5]),
       'derniere_colonne': array([10, 11, 12, 13, 14, 15]),
       'premier_ligne': array([ 0,  2,  4,  6,  8, 10]),
       'toute_matrice': array([[ 0,  2,  4,  6,  8, 10],
                               [ 1,  3,  5,  7,  9, 11],
                               [ 2,  4,  6,  8, 10, 12],
                               [ 3,  5,  7,  9, 11, 13],
                               [ 4,  6,  8, 10, 12, 14],
                               [ 5,  7,  9, 11, 13, 15]]),
       'colonne_deux_avant_derniere': array([[ 2,  4,  6,  8],
                                             [ 3,  5,  7,  9],
                                             [ 4,  6,  8, 10],
                                             [ 5,  7,  9, 11],
                                             [ 6,  8, 10, 12],
                                             [ 7,  9, 11, 13]])}
```

Quelques exemples introductifs à NumPy (Opérations Matricielles)

Opérations élémentaires

```
[10]: # Création de trois matrices 2d
      # Création d'un matrice A(4*3)
      A = np.array(
          [[10, 12, 14],
           [1, 2, 3],
           [0, 1, 4],
           [0, 5, 7]]
      )
      # Création d'un matrice B(4*3)
      B = np.array(
          [[1, 2, 4],
           [1, 1, 3],
           [0, 2, 4],
           [0, 0, 3]]
      )
      # Création d'un matrice C(3*4)
      C = np.array(
          [[1, 2, 4, 0],
           [1, 1, 3, 2],
           [0, 2, 4, 1]]
      )
      D = np.add(A, B)  # Somme des deux matrices à l'aide de la fonction add de NumPy
      E = A + B  # Somme des deux matrices à l'aide de l'opérateur "+"
      F = 10*A  # Calcul du produit par un scalaire de la matrice A
      G = np.dot(A, C)  # Calcul du produit matriciel A*C à l'aide de .dot numpy
      H = C@A   # Calcul du produit matriciel C*A à l'aide de l'opérateur @
      # Affichage des résultats à partir d'un dictionnaire
      dict(
          somme_numpy=D,
          somme_operateur=E,
          produit_scalaire=F,
          produit_matriciel_AC=G,
          produit_matriciel_CA=H
      )
```

```
[10]: {'somme_numpy': array([[11, 14, 18],
              [ 2,  3,  6],
              [ 0,  3,  8],
              [ 0,  5, 10]]),
       'somme_operateur': array([[11, 14, 18],
              [ 2,  3,  6],
              [ 0,  3,  8],
              [ 0,  5, 10]]),
       'produit_scalaire': array([[100, 120, 140],
              [ 10,  20,  30],
              [  0,  10,  40],
              [  0,  50,  70]]),
       'produit_matriciel_AC': array([[ 22,  60, 132,  38],
              [  3,  10,  22,   7],
              [  1,   9,  19,   6],
              [  5,  19,  43,  17]]),
       'produit_matriciel_CA': array([[12, 20, 36],
              [11, 27, 43],
              [ 2, 13, 29]])}
```

Résultats égaux np.dot = @

A*C != C*A le nombre de colonnes de la première matrice doit être le même au nombre de lignes de la deuxième matrice

Quelques opérations d'algèbre linéaire

```
[11]: # Création de deux matrices. Une matrice carrée MCAR (4*4) et une matrice non carrée MNCAR (3*4)
      MCAR = np.array(
          [[1, 2, 1, 3],
           [1, 0, 1, 2],
           [2, 1, 2, 1],
           [3, 4, 6, 2]]
      )

      MNCAR =  np.array(
          [[1, 2, 1, 3],
           [1, 0, 1, 2],
           [2, 1, 2, 1]]
      )
      # Application d'opérations linalg sur MCAR
      det_mcar = np.linalg.det(MCAR)  # Calcul du détetminant d la matrice MCAR
      inv_mcar = np.linalg.inv(MCAR)  # Calcul de la matrice inverse de MCAR
      # Application d'opérations linalg sur MNCAR
      trans_mncar = MNCAR.T  # Transposition de la matrice MNCAR
      racine_elem_elem_MNCAR = np.sqrt(MNCAR)  # Racine carrée élément par élément de la matrice MNCAR


      # Création du dictionnaire pour affichage des résultats
      dict(
          det_mcar=det_mcar,
          inv_mcar=inv_mcar,
          trans_mncar=trans_mncar,
          racine_elem_elem_MNCAR=racine_elem_elem_MNCAR
      )
```

```
[11]: {'det_mcar': -21.000000000000001,
       'inv_mcar': array([[ 0.0952381 , -0.38095238,  1.14285714, -0.33333333],
              [ 0.42857143, -0.71428571,  0.14285714, -0.        ],
              [-0.38095238,  0.52380952, -0.57142857,  0.33333333],
              [ 0.14285714,  0.42857143, -0.28571429,  0.        ]]),
       'trans_mncar': array([[1, 1, 2],
              [2, 0, 1],
              [1, 1, 2],
              [3, 2, 1]]),
       'racine_elem_elem_MNCAR': array([[1.        , 1.41421356, 1.        , 1.73205081],
              [1.        , 0.        , 1.        , 1.41421356],
              [1.41421356, 1.        , 1.41421356, 1.        ]])}
```