TP 1 – Tableaux de données multidimensionnels avec NumPy

S.Gibet

Année 2022-2023

Les packages au coeur de l'informatique scientifique et des sciences pour l'ingénieur sont les suivants :

- SciPy: librairie fondamentale pour le calcul scientifique
- NumPy : package "array" multidimensionnel
- Matplotlib: 2D plotting
- Sympy: symbolic mathematics
- IPython: enhanced interactive console
- pandas: structures de données et analyse

Les exercices de ce TP ont pour objectif de se familiariser avec NumPy pour le traitement de données vectorielles et matricielles en Python. La documentation officielle de Python se trouve à l'url : `https://docs.python.org/3/tutorial/`. Pour l'installation de packages aller à : `https://scipy.org/install.html`. Pour une doc d'initialisation à NumPy aller à : `https://docs.scipy.org/doc/numpy/user/quickstart.html`.
Ce TP est divisé en trois parties : la première partie (questions 1 et 2) est à réaliser le 7 septembre 2021. La seconde partie est à réaliser et rendre le 14 septembre 2021. Le problème sera terminé à la maison et devra être rendu le

Configuration de l'environnement de programmation

Installer Anaconda et spyder – Anaconda et *Spyder* est l'environnement de développement que nous allons utiliser pour écrire nos programmes. Python 3.9 et Anaconda 2.2.2 ont été installés dans les salles informatique de l'UBS.

IDE Spyder – La fenêtre Spyder est divisée en plusieurs zones, de haut en bas :

- la barre de menu permet d'avoir accès aux principales commandes,
- la barre d'outils montre les outils les plus courants : Nouveau fichier, Ouvrir un fichier, Exécuter le fichier, ...
- la zone d'édition à gauche contient le programme à éditer, c-à-d. l'endroit où vous allez écrire et modifier vos programmes Python
- la zone en bas à droite (dans le 1/4 inférieur droit) est la console Python. Elle permet de :
  - saisir directement des instructions Python

    * afficher les messages d'erreur (à l'interprétation de votre programme)
    * afficher les traces des exécutions de vos programmes (fonction print)
    * saisir les entrées de votre programme (fonction input)
  - l'explorateur de variables dans le quart supérieur droit

Partie I

1 Le tableau NumPy

ndarray– NumPy permet de manipuler des tableaux multidimensionnels (vecteurs, matrices) pour le calcul scientifique en Python. Dans cette section vous explorez rapidement la représentation des données vectorielles (vecteurs et matrices), leur création, manipulation et leurs propriétés. Vous utiliserez les modules `numpy` et `numpy.linalg`.

1.1 Performance avec NumPy

Soit le programme suivant qui utilise, soit des listes (1ère partie), soit des arrays (2nde partie). Observez les temps d'exécution.

```
import numpy as np
from random import random
from operator import add
import time

n = int(input("Entrez un entier : "))
l1 = [random() for i in range(n)]
l2 = [random() for i in range(n)]

start = time.perf_counter()
l3 = map(add, l1, l2)
end = time.perf_counter()
print( end - start)

A1 = np.array(l1)
A2 = np.array(l2)

start = time.perf_counter()
A3 = A1 + A2
end = time.perf_counter()
print( end - start)
```

1.2 Création d'arrays numpy

- à partie de listes ou n-uplets Python
- en utilisant des fonctions dédiées, telles que *arange*, *linspace*, etc.
- par chargement à partir de fichiers

1. Ecrivez le programme suivant qui crée un vecteur contenant des entiers et permet de visualiser ces données.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array([0,1,2,3])
v = np.array([1,3,2,4])
print(v)
print(type(v))
fig = plt.figure()
plt.plot(x,v,'rv--', label = 'v(x)')
plt.legend(loc='lower right')
plt.xlabel('x')
plt.ylabel('v')
plt.title('Mon titre')
plt.xlim([-1,4])
plt.ylim([0,5])
plt.show()
fig.savefig('toto.png')
```

2. Créez un vecteur v contenant des réels de dimension 1, une matrice M contenant des entiers de dimension 2x2 (2 lignes, 2 colonnes).

3. Pour récupérer les types des éléments des arrays, vous utiliserez dtype ($v.dtype$ ou $M.dtype$). Attention, les types doivent être respectés lors d'assignations à des arrays.
   Que se passe-t-il si vous faites : M[0,0] = "hello" ?
   De même, vérifiez ce que la séquence d'instructions suivantes réalise.

   ```
   a=np.array([1,2,3])
   a[0] = 3.2
   print(a)
   a.dtype
   ```

   On peut définir de manière explicite le type des données en utilisant le mot clé *dtype* en argument :

   ```
   M = np.array([[1,2],[3,4]], dtype = float)
   print(M)
   M = np.array([[1,2],[3,4]], dtype = int)
   print(M)
   M = np.array([[1,2],[3,4]], dtype = complex)
   print(M)
   ```

4. Vérifiez le type des données du vecteur v et de la matrice M (fonction *type*). Donnez la dimension de v et de M à l'aide de shape, que vous utiliserez, soit par $v.shape$ (resp. $M.shape$), soit par $np.shape(v)$ (resp. $np.shape(M)$).

5. Génération d'arrays : créez un tableau contenant les valeurs de 0 à 9 en utilisant les fonctions *arange* (create a range), *linspace* (début et fin inclus) et *logspace*. Vous regarderez la documentation de ces fonctions.

6. Tirages aléatoires. Créez une matrice M de dimension (3,3) à partir d'un tirage uniforme dans [0,1] (vous inclurez "from numpy import random"), puis suivant une loi normale standard (np.random.randn(3,3)).
   Pour vérifier la loi normale vous afficherez l'histogramme des tirages : a = random.randn(10000), hist = plt.hist(a,40)

7. Matrice contenant des zéros ou des uns. Créez un vecteur contenant 3 zéros ("np.zéros"), puis une matrice 3x3 contenant des uns ("np.ones").

8. Créez une matrice diagonale contenant les éléments 1, 2, 3 sur la diagonale (*np.diag* en passant une liste en paramètres).Vous pourrez également tester la fonction *np.identity* (en passant la dimension de la diagonale et le type des éléments).

9. Créez une matrice (3,5) contenant des valeurs infinies (nan).

10. Fonction des indices. Soit la fonction suivante:

    ```
    def initfunction(i,j):
        return 100 + 10*i + j
    ```

    Créez une matrice de 5 lignes et 3 colonnes qui utilise les valeurs définies par la fonction *initfunction* ci-dessus, en utilisant la fonction *fromfunction*.

11. A partir d'un fichier. Créez une matrice de uns de dimension (3 x 5 x 7). A l'aide de la fonction *save*, sauvegardez la matrice dans le fichier "file.npy", et chargez la matrice b à l'aide de la fonction *load* appliquée sur ce fichier.

2 Manipulations d'arrays

1. Indexation : pour un vecteur, il n'y a qu'une seule dimension, d'où un seul indice (v[0]). Pour un array à 2 dimensions on a 2 indices (M[0,0], M[1,1]) (ou bien M[0][0], M[1][1]), pour un array à 3 dimensions 3 indices, etc. Créez 2 matrices A et B de tailles 4x2 et 2x3. Vérifiez les accès aux différents éléments, ainsi que la forme des matrices (shape) et la dimension (nombre d'éléments).

2. Reprenez la matrice M. Donnez le résultat de la 2ème ligne d'indice 1, de la 2ème colonne d'indice 1. Vous vérifierez qu'on peut assigner de nouveaux éléments dans les matrices, et de nouvelles lignes ou colonnes.

3. Slicing ou accès par tranches. Pour accéder à des tranches de matrices, vous utiliserez la syntaxe M[start:stop:step]. Attention : on commence à l'indice start (compris) et on s'arrête à l'indice stop, par pas de step.

Partie II

3 Traitement de données vectorielles et matricielles

Dans cet exercice, afin de vous entraîner à écrire des fonctions en Python, vous écrirez les opérations demandées (i) **sans** la librairie numpy (fonctions avec boucles), (ii) puis vous utiliserez les fonctions de NumPy quand elles existent.

1. Addition de matrices. Créez 2 matrices A et B de tailles 3x2. Faites l'addition des 2 matrices (vous pourrez utiliser directement A + B avec NumPy).

2. Multiplication scalaire. Multipliez tous les éléments d'une matrice par 3 et divisez tous les éléments de l'autre matrice par 4.

3. Testez la combinaison linéaire de vecteurs : par exemple 2*v -3*w + u/3, u, v, et w étant des vecteurs.

4. Multiplication de matrice.

   - Ecrivez l'algorithme qui prend en argument deux matrices et retourne la matrice produit.
   - Utilisez la fonction $np.dot(A, B)$.

   Vérifiez ce que donne le produit A . x, A étant une matrice 3x2 et x un vecteur 2x1.
   Vérifiez sur des exemples choisis que l'on peut faire des produits de matrices A . B, avec A (m,n) et B (n,m).

5. Effectuez le produit de plusieurs matrices à l'aide de la fonction *dot*.

6. Ecrivez la fonction qui permet de calculer des puissances de matrices. Vous testerez ensuite la fonction *matrix_power* du module *numpy.linalg*.

7. Vérifiez que AxB n'est pas égal à BxA (le produit de matrices n'est pas commutatif)

8. Vérifiez que (AxB)xC est égale à Ax(BxC) (le produit de matrices est associatif)

9. Matrice identité. Vérifiez les propriétés de la matrice identité I (A x I = I * A = A)

10. Matrice transpose. Déterminez la matrice transpose d'une matrice A. Vous utiliserez la notation A.T. Vous pouvez aussi utiliser la fonction *transpose* (np.transpose(A)). Prenez des exemples pour tester.

11. Le déterminant, le rang et la trace d'une matrice s'obtiennent par les fonctions *det*, *matrix_rank* du module *numpy.linalg* et *trace* du module *numpy*. Enfin la fonction *inv* du module *numpy.linalg* renvoie l'inverse de la matrice s'il existe. Testez ces fonctions sur la matrice A = np.array([[1,2], [3,4]]).

12. Pour résoudre le système linéaire $Ax = b$, lorsque la matrice A est inversible, on peut employer la fonction *solve* du module *numpy.linalg*. Prenez l'exemple avec b = np.array([1,5])

13. Ecrivez la fonction qui effectue un produit scalaire entre deux vecteurs. Vous utiliserez cette fonction avec 2 vecteurs de dimension 10. Faites attention à bien utiliser le produit d'un vecteur de dimension 1 * 10 par un vecteur de dimension 10 * 1 de façon à obtenir un résultat scalaire. Utilisez la fonction *vdot* qui permet de calculer le produit scalaire de 2 vecteurs. Testez.

14. Ecrivez la fonction qui effectue le produit vectoriel de 2 vecteurs. Utilisez la fonction *cross* qui réalise ce produit vectoriel. Testez.

15. Ecrivez la fonction Python qui calcule la norme Euclidienne d'un vecteur de dimension quelconque.

Partie III

4 Problème

Calculer les valeurs propres et les vecteurs propres d'une matrice est un problème que l'on trouve fréquemment dans le cadre de la résolution de systèmes linéaires. Cela permet en particulier de réduire la dimension d'un signal. Par exemple l'analyse en composantes principales (ACP) exploite le calcul de vecteurs propres. Cette technique peut conduire à la compression d'images ou de séries temporelles.

Vous allez implémenter dans ce petit problème un algorithme qui utilise la méthode des puissances pour déterminer la valeur propre dominante réelle d'une matrice carrée ainsi que le vecteur propre principal. L'idée repose sur le fait que la suite :

$$\frac{x}{\|x\|}, \frac{Ax}{\|Ax\|}, \frac{A^2x}{\|A^2x\|}, ... \tag{1}$$

converge, dans certaines conditions, vers le vecteur propre principal associé à la valeur propre maximale en valeur absolue.

L'algorithme est un algorithme d'optimisation qui doit converger au bout d'un nombre fini (pas trop grand) d'itérations. Il est donné ci-dessous.

Input: une matrice A carrée avec une valeur propre dominante réelle ;
$v^{(0)}$ un vecteur de norme 1.
Output: Un vecteur propre v et une valeur propre λ
for k = 1 to n do
    $w = Av^{(k-1)}$
    $v^{(k)} = w/\|w\|$
    $\lambda^{(k)} =^t v^{(k)} A v^{(k)}$
end
Attention : $v^{(k)}$ et $\lambda^{(k)}$ signifient respectivement le vecteur v et le vecteur λ à l'itération k de l'algorithme.

1. Écrivez cet algorithme en Python et testez le sur un cas de matrice carrée A donnée. Pour que l'algorithme converge, il faut que la matrice A possède une valeur propre dominante réelle. De plus, le vecteur initial $v^{(0)}$ ne doit pas être orthogonal au vecteur propre recherché.

2. Vérifiez les résultats précédents en utilisant les fonctions de la librairie `numpy.linalg.eig`.