Informatique - Notes de cours 9

Manipulations des tableaux `Numpy`

M. Marmorat, M. Morel

12 février 2025

`numpy` est une bibliothéque destinée à manipuler des tableaux (multidimensionnels) ainsi que les fonctions mathématiques qui opèrent sur ces tableaux.
Dans ce cours, nous considérerons les tableaux bidimensionnels dans le but de représenter des matrices, des images , etc...
Comment charger la bibliothèque `numpy` ?

1 Création

Un tableau est un ensemble d'éléments de **méme type**, organisés en lignes et en colonnnes.

Syntaxe :
Pour créer un tableau bidimensionnel, on donne ses éléments sous forme d'une liste de listes , que l'on transforme en tableau en lui appliquant la fonction `np.array`.
Pour définir le tableau suivant :

x_{11}	x_{12}	$\cdots$	x_{1m}
x_{21}	x_{22}	$\cdots$	x_{2m}
$\vdots$	$\vdots$		$\vdots$
x_{n1}	x_{n2}	$\cdots$	x_{nm}

voici la syntaxe :

```
tab=np.array([x_11,x_12,..., x_1m], [x_21,x_22,..., x_2m], ..., [x_n1,
   x_n2,..., x_nm]])
```

Chaque ligne est définie comme une liste ; ces listes sont réunies dans une liste de listes. On applique ensuite la fonction `np.array`, qui renvoie un tableau numpy.

Par exemple

```
A = np.array([[0, 1], [1, 0]])
```

crée la matrice $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$.

Remarque 1. Il y a une grande différence entre les tableaux et les listes : les listes peuvent contenir des objets de plusieurs types, mais pas les tableaux. On rencontrera donc des tableaux d'entiers, de flottants (permettant de faire du calcul matriciel) ; de booléens ; de chaénes de caractéres (permettant de programmer des jeux).

Exercice 1. 1. Stocker la matrice $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ dans un tableau numpy.

2. Idem pour la matrice nulle de $\mathcal{M}_{2,4}(\mathbb{K})$.
3. Idem la matrice identité de $\mathcal{M}_3(\mathbb{K})$.

Exercice 2. Pour créer un tableau de taille $n \times m$ rempli intégralement de la valeur 0 :

```
Z=np.array(n*[m*[0]]) ou Z=np.zeros((n,m))
```

Exercice 3. Pour créer un tableau de taille $n \times m$ dont tous les éléments sont des nombres aléatoires de l'intervalle $]0, 1[$, on utilise la syntaxe suivante :

```
H=np.rand(n,m)
```

2 Accès aux éléments

On se donne un tableau T de taille $n \times m$, et i, j deux entiers appartenant respectivement é $[\![0, n-1]\!]$ et $[\![0, m-1]\!]$.

Instruction Python	Opérations sur les tableaux
`np.shape(T)`	donne les dimensions de T sous la forme (n, m)
`np.size(T)`	donne le nombre d'éléments de T
`np.size(T,0)`	donne le nombre de lignes de T
`np.size(T,1)`	donne le nombre de colonnes de T
`T[i,j]`	renvoie l'élément de T situé sur la i-ème ligne et j-ème colonne
`T[i,:]`	renvoie la i-ème ligne de T
`T[:,j]`	renvoie la j-ème colonne de T

Remarque 2. Comme pour les listes, la numérotation des indices commence à 0 et s'arrête à $n-1$ pour les lignes et $m-1$ pour les colonnes.
Les indices négatifs permettent de parcourir les numéros de ligne ou colonne en partant de la fin.

Remarque 3 (Rappel)**.** Il est possible de modifier (ou d'effacer) un élément d'une liste au sein même de cette liste. On dit que les listes sont mutables.
La mutabilité des listes oblige à la prudence quand on souhaite dupliquer une liste. Par exemple, si on tape dans la console :

```
x=[5,2,9] ; y=x
```

on dispose en fait d'une seule et méme liste qui posséde deux noms. Si l'on écrit alors dans la console :

```
x[2]=10 ; x ; y
```

la liste devient mais son adresse ne change pas en mémoire. Les variables `x` et `y` pointent donc vers la même liste, et ont donc la même valeur

Tout se passe donc comme si la mutation de `x` s'était répercutée sur `y`.

La syntaxe pour dupliquer la liste , c'est-à-dire créer une autre variable de même valeur que `x` , est :

```
y=x[:]
```

Remarque 4. Pour économiser de l'espace mémoire, `Python` évite autant que possible de copier des listes ou des tableaux. Ainsi, si la variable `a` désigne une liste ou un tableau, l'instruction `b=a` ne crée pas une nouvelle

copie de a, elle se contente d'attribuer un nouveau nom à la liste ou au tableau. Dés lors, toute modification de a affecte b et vice-versa. Il n'est donc pas possible, par simple affectation, de dupliquer une liste ou un tableau.

Pour les listes, on rappelle que l'instruction L_bis = L[:] crée une vraie copie de L. Si l'on procède ensuite à une modification de Lbis, cela ne change pas L et vice-versa.

Pour les tableaux : si tab désigne un tableau donné, l'instruction tab_bis=tab[:,:] crée une copie de tab et la stocke dans tab_bis.

3 Opérations sur les tableaux

3.1 Opérations élémentaires

Instruction Python	Opérations sur les tableaux
in	appartenance
sum	somme

Exercice 4. On donne les instructions suivantes :

```
T=np.array([[1,2,3],[4,5,6]])
1 in T
0 in T
s=sum(T)
print(s)
```

Que renvoie ce script ?

Remarque 5. Attention : le test d'égalité == ne permet pas, comme pour les listes, de comparer globalement deux tableaux.

Si les tableaux sont de même taille, la comparaison se fait case par case : on obtient donc un tableau de booléens et non un seul booléen ; sinon, l'ordinateur renvoie False.

3.2 Opérations arithmétiques

Contrairement aux listes, on peut effectuer les opérations mathématiques (addition, multiplication, division, soustraction) sur les tableaux. Toutes ces opérations agissent case par case entre tableaux de méme format.
Syntaxe :

```
S+T
S-T
S*T
S/T
a*T
```

Remarque 6. L'addition et la multiplication par un float (dilatation) correspondent aux opérations connues sur les matrices, mais la multiplication ne correspond pas du tout au produit matriciel ! Il s'agit d'une opération coefficient par coefficient entre deux tableaux de même taille.

Exercice 5. Effectuer les opérations entre les deux tableaux suivants :

```
T=np.array([[1,2,3],[4,5,6]])
S=np.array([[0,0,7],[6,6,6]])
```

Remarque 7. Concernant l'addition et la dilatation, il y a une grosse différence entre listes et tableaux.

En effet, pour les listes, $+$ correspond à une concaténation. Pour les tableaux `numpy`, c'est une simple addition case par case.

L'opération $n*$ crée une nouvelle liste constituée de n copies de la liste de départ. Mais pour les tableaux, chaque coefficient du tableau de départ est multiplié par n.

3.3 Opérations fonctionnelles

Toutes les fonctions mathématiques de référence peuvent être utilisées sur les tableaux, si elles proviennent de la bibliothèque `numpy`. Elles agissent coefficient par coefficient sur les tableaux.

Syntaxe : on conserve la notation habituelle d'une fonction, sauf que la variable est un tableau.

Exercice 6. :

```
T=np.array([[1,2,3],[4,5,6]])
U=T**2
A=np.exp(T); B=np.sin(T)
```

3.4 Produit de matrices

Le produit de matrices s'obtient en Python grâce à la commande `np.dot`.

Exercice 7.

1. Si $A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{pmatrix}$, calculer à la main ce que vaut A^2.
2. Stocker la matrice A dans un tableau numpy `A`, et calculer A^2 grâce à l'instruction

```
np.dot(A,A)
```

3.5 Transposée d'une matrice

La transposée d'une matrice peut s'obtenir par la fonction `np.transpose`.

Exercice 8.

1. Testez par exemple les lignes suivantes

```
A = np.random.rand(3, 4)
print("A = ", A)

B = np.transpose(A)
print("B = ", B)
```

   Si $A \in \mathcal{M}_{m,n}(\mathbb{K})$, quelles sont les dimensions de ${}^tA$?
2. Implémenter une fonction `transposition` qui prenne en argument une matrice A (sous la forme d'un tableau), et qui calcule sa transposée (sans utiliser la fonction numpy bien entendu).

4 Exercices

Exercice 9. Tester les instructions suivantes

```
B = np.eye(4)
C = np.ones((4,3))
D = np.zeros((5,2))
```

```
E = np.diag([1,2,3,4]
```

Que fait chacune d'entre elle ?

Exercice 10. 1. Écrire une fonction `plus` prenant en argument un nombre x et un tableau numpy T, et qui ajoute x à l'élément en haut à gauche du tableau T. Vérifiez le comportement de votre fonction sur l'exemple suivant

```
T = np.array([-1, 4, 3], [2, 3, -12])
plus(6, T)
print(T)
```

2. Écrire une fonction `top_right` prenant en argument un entier naturel non nul n et renvoyant une matrice carrée de taille n dont les coefficients sont nuls sauf celui en haut à droite qui vaut 1. Par exemple l'insctruction `top_right(4)` doit renvoyer $\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$.

Exercice 11 (Quelques matrices particulières)**.** Soit $M = (m_{ij}) \in \mathcal{M}_n(\mathbb{R})$ une matrice carrée fixée.

1. Rappeler la définition mathématique de l'assertion "M est une matrice diagonale".
2. Écrire une fonction Python `est_diagonale` qui prend en argument un tableau Numpy et qui renvoie `True` si la matrice associée est diagonale et `False` sinon (on prendra soin de vérifier si la matrice est carrée !). Testez votre fonction.
3. Rappeler la définition mathématique de l'assertion "M est une matrice triangulaire supérieure".
4. Écrire une fonction Python `est_triangulaire_superieure` qui prend en argument un tableau Numpy et qui renvoie `True` si la matrice associée est triangulaire supérieure et `False` sinon (on prendra soin de vérifier si la matrice est carrée !). Testez votre fonction.
5. Rappeler la définition mathématique de l'assertion "M est une matrice symétrique".
6. Écrire une fonction Python `est_symetrique` qui prend en argument un tableau Numpy et qui renvoie `True` si la matrice associée est symétrique et `False` sinon (on prendra soin de vérifier si la matrice est carrée !). Testez votre fonction.

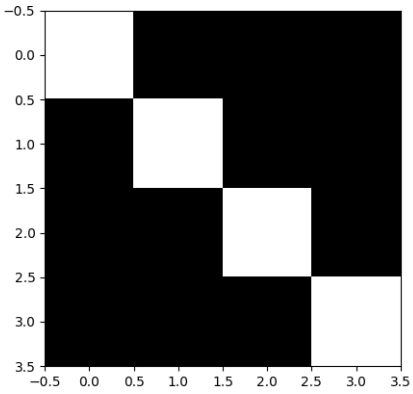
Exercice 12. 1. Écrire une fonction `trace` renvoyant la somme des coefficients diagonaux d'une matrice carrée.

2. Écrire une fonction `somme` renvoyant la somme de tous les coefficients d'une matrice quelconque.
3. En utilisant la fonction précédente, écrire une fonction `moyenne` renvoyant la moyenne de tous les coefficients d'une matrice quelconque.

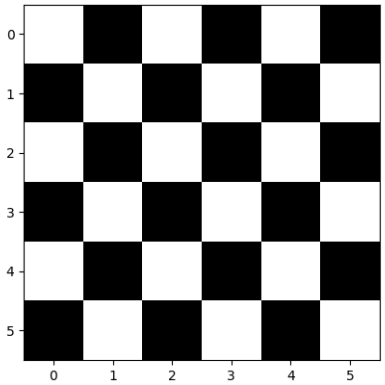
Exercice 13. Les matrices sont un moyen de coder des images en Python. Chaque coefficient de la matrice correspond à un pixel et indique la couleur du pixel. Dans cet exercice on se limite à des images en noir et blanc, on utilise pour cela des coefficients compris entre 0 et 1 avec la convention : 0 = noir, 1 = blanc (et toute valeur comprise entre 0 et 1 correspond à une nuance de gris). Voici un exemple de correspondance entre matrice et image : les instructions

```
A = np.eye(4)
plt.imshow(A, cmap="gray")
```

donnent l'image



Écrire une fonction `damier` prenant en argument un entier n et renvoyant un damier de taille $n \times n$. Par exemple, `damier(6)` doit renvoyer l'image ci-dessous



Exercice 14.

1. En utilisant le produit matriciel implémenté par numpy, écrire une fonction puissance prenant en arguments une matrice carrée M et un entier n, et renvoyant M^n. Tester sur un exemple.
2. Écrire une fonction `commute` prenant en entrée deux matrices carrées de même taille et renvoyant `True` si elles commutent et `False` sinon. On testera la fonction sur les matrices :

```
A1 = np.eye(3)
A2 = np.ones((3,3))
A3 = np.diag([1,2,3])
```