

# Algorithmes avec boucles imbriquées

Méthodes de programmation

## 1 Bref rappel sur les tableaux à deux dimensions

Le code ci-dessous est accessible en ligne dans le fichier `Intro.py` :

```

1  A = [[ 1. , 2. ],
      [ 1.5 , 2.5],
3   [ 2. , 3. ],
      [ 2.6 , 3.6]]
5  print (type(A))
  A=np.array(A) ; print(type(A))
7  dim_A=A.shape ; print(dim_A) ; print(type(dim_A))
  print (A[0]);print(A[1]);print(type(A[0]))
9  print (A[0,0]);print(A[0,1]);print(type(A[0,0]))
  print (A[:,0]);print(A[:,1]);print(type(A[:,0]))
11 print (A[0,:]);print(A[2,:]);

13 Un=np.ones([2,2]) ; print(Un)
  Ze=np.zeros([2,2]) ; print(Ze)
15
  dim_A=A.shape ; print(dim_A) ; print(type(dim_A))

```

```

<class 'list'>
<class 'numpy.ndarray'>

```

```

(4, 2)
<class 'tuple'>

```

```

[ 1.  2.]
[ 1.5  2.5]
<class 'numpy.ndarray'>

```

```

1.0
2.0
<class 'numpy.float64'>

```

```

[ 1.   1.5  2.   2.6]
[ 2.   2.5  3.   3.6]

```

```

<class 'numpy.ndarray'>
[ 1.  2.]
[ 2.  3.]

```

```

[[ 1.  1.]
 [ 1.  1.]]

```

```

[[ 0.  0.]
 [ 0.  0.]]

```

```

(4, 2)
<class 'tuple'>

```

## 2 Les boucles imbriquées dans le calcul matriciel

### 2.1 Transposée d'une matrice

Pour rappel, la transposition d'une matrice consiste à inverser les lignes avec les colonnes. Ainsi, on a, sur deux exemples :

$$M_1 = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \Rightarrow {}^t M_1 = \begin{pmatrix} a & c \\ b & d \end{pmatrix} \quad ; \quad M_2 = \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix} \Rightarrow {}^t M_2 = \begin{pmatrix} a & d \\ b & e \\ c & f \end{pmatrix}$$

#### Activité 1 Ecriture d'une fonction

**nom :** transposer  
**arguments :** m (array)  
**effet :** Renvoie le tableau (matrice) transposé

Utiliser la fonction avec l'exemple suivant :

```

2 M = np.array([[1, 2, 3], [4, 5, 6]])
3
4 N=transposer(M)
5 dim_N=N.shape ; print(dim_N) ; print(N)
6 N2=np.transpose(M) ; print(N2) # on utilise une fonction préimplantée dans Numpy, pour vérification.
```

### 2.2 Somme de deux matrices

- On ne peut sommer que des matrices de même dimension.
- Les matrices à sommer ne sont pas nécessairement carrées.
- La somme de deux matrices est une opération commutative.

#### Activité 2 Ecriture d'une fonction

**nom :** somme\_matrices  
**arguments :** m,n (array,array)  
**effet :** Renvoie le tableau correspondant à la somme des deux matrices.

Utiliser la fonction avec l'exemple suivant :

```

2 M = np.array([[1, 2, 3], [4, 5, 6],[1, 2, 3]])
3 N = np.array([[2, 7, 1], [3, 9, 1],[2, 4, 6]])
4 P=somme_matrices(M,N) ; print(P)
5 P2=M+N ; print(P2) # permet de vérifier le résultat
```

### 2.3 Produit de deux matrices

- Le produit de deux matrices ne peut se définir que si le nombre de colonnes de la première matrice est le même que le nombre de lignes de la deuxième matrice, c'est-à-dire lorsqu'elles sont de type compatible.
- Les matrices à multiplier ne sont pas nécessairement carrées.
- Le produit de deux matrices n'est pas une opération commutative.

Si  $A = (a_{ij})$  est une matrice de type  $(m, n)$  et  $B = (b_{ij})$  est une matrice de type  $(n, p)$ , alors leur produit, noté  $AB = (c_{ij})$  est une matrice de type  $(m, p)$  donnée par :

$$\forall i, j : c_{ij} = \sum_{k=1}^n a_{ik}b_{kj} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$$

$$\text{Exemple : } \begin{pmatrix} 1 & 0 \\ 2 & -1 \end{pmatrix} \times \begin{pmatrix} 3 & 4 \\ -2 & -3 \end{pmatrix} = \begin{pmatrix} (1 \times 3 + 0 \times -2) & (1 \times 4 + 0 \times -3) \\ (2 \times 3 + -1 \times -2) & (2 \times 4 + -1 \times -3) \end{pmatrix} = \begin{pmatrix} 3 & 4 \\ 8 & 11 \end{pmatrix}$$

On peut proposer une façon plus visuelle d'effectuer les calculs. On dispose comme ci-dessous les matrices à multiplier.

Voici comment obtenir l'élément  $c_{11}$  : on multiplie les éléments encadrés en traits pleins, on multiplie les éléments encadrés en traits pointillés, et on ajoute les deux résultats. L'avantage de cette méthode visuelle est que l'on ne peut pas se tromper de ligne ou de colonne pour obtenir l'élément  $c_{ij}$ .

$$\begin{pmatrix} \boxed{3} & 4 \\ \text{---}\boxed{-2}\text{---} & -3 \end{pmatrix} \\ \begin{pmatrix} \boxed{1} & \text{---}\boxed{0}\text{---} \\ 2 & -1 \end{pmatrix} \quad \begin{pmatrix} \boxed{3} & 4 \\ 8 & 11 \end{pmatrix}$$

Voici comment obtenir l'élément  $c_{12}$  :

$$\begin{pmatrix} 3 & \boxed{4} \\ -2 & \text{---}\boxed{-3}\text{---} \end{pmatrix} \\ \begin{pmatrix} \boxed{1} & \text{---}\boxed{0}\text{---} \\ 2 & -1 \end{pmatrix} \quad \begin{pmatrix} 3 & \boxed{4} \\ 8 & 11 \end{pmatrix}$$

En général, comme rappelé précédemment,  $AB$  n'est pas égal à  $BA$ , comme le montre l'exemple suivant :

$$\begin{pmatrix} 1 & 2 & 0 \\ 4 & 3 & -1 \end{pmatrix} \begin{pmatrix} 5 & 1 \\ 2 & 3 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 9 & 7 \\ 23 & 9 \end{pmatrix} \quad \text{tandis que} \quad \begin{pmatrix} 5 & 1 \\ 2 & 3 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 1 & 2 & 0 \\ 4 & 3 & -1 \end{pmatrix} = \begin{pmatrix} 9 & 13 & -1 \\ 14 & 13 & -3 \\ 19 & 18 & -4 \end{pmatrix}$$

### Activité 3 Ecriture d'une fonction

**nom :** produit\_matrices

**arguments :** m,n (array,array)

**effet :** Renvoie le tableau correspondant au produit des deux matrices.

Utiliser la fonction avec l'exemple suivant :

```
1 T = np.array([[1, 2], [3,4], [5,6]])
  U = np.array([[2, 7, 6, 9], [3, 9, 10, 2]])
3
  V=produit_matrices(T,U) ; print(V)
5 V2=np.dot(T,U) ; print(V2) # permet de vérifier le résultat
```

Peut-on ici vérifier que le produit de deux matrices n'est pas commutatif? Sinon, proposez un exemple qui permette d'illustrer que le produit de deux matrices n'est pas commutatif.

## 3 Le slicing à la rescousse

Reprendre la programmation des trois problématiques précédentes en utilisation la possibilité de faire du slicing.

Pour la multiplication de deux matrices, vous aurez besoin de savoir que si  $V$  est un vecteur (array de dimension 1), alors `np.sum(V)` renvoie la somme de toutes les coordonnées de  $V$ .

Par cette nouvelle programmation, vous montrerez ainsi que l'on économise une boucle `for` à chaque fois. Ainsi, le temps de calcul est grandement diminué. Par exemple, pour la multiplication de deux matrices 300 par 300, le calcul est effectué environ 20 fois plus rapidement avec une programmation utilisant le slicing.