Les tableaux avec numpy

Lycée Pierre Corneille – MP

2016-2017

Structure de tableau

- Constituants repérés par un tuple

Structure de tableau

- Constituants repérés par un tuple
- **Format** : tuple d'entiers `A.shape`

Structure de tableau

- Constituants repérés par un tuple
- **Format** : tuple d'entiers `A.shape`
- **Dimension** : longueur du format

Structure de tableau

- Constituants repérés par un tuple
- **Format** : tuple d'entiers `A.shape`
- **Dimension** : longueur du format
- **Taille** : nombre total d'éléments `A.size`

Structure de tableau

- Constituants repérés par un tuple
- **Format** : tuple d'entiers `A.shape`
- **Dimension** : longueur du format
- **Taille** : nombre total d'éléments `A.size`

Les formats `f`$=(f_0,\dots,f_{r-1})$ et `t`$=(t_0,\dots,t_{d-1})$ sont **compatibles** lorsqu'ils représentent des tableaux de même taille :

$$f_0 \times \cdots \times f_{r-1} = t_0 \times \cdots \times t_{d-1}.$$

Structure de tableau

Tableau unidimensionnel

Structure de tableau

Tableau unidimensionnel

- `A.shape` $(n,)$ liste

Structure de tableau

Tableau unidimensionnel

- `A.shape` $(n,)$ liste

Tableau bidimensionnel

Structure de tableau

Tableau unidimensionnel

- `A.shape` $(n,)$ liste

Tableau bidimensionnel

- `A.shape` (n,p) matrice $A \in \mathfrak{M}_{n,p}(\mathbb{K})$

Structure de tableau

Tableau unidimensionnel

- `A.shape` $(n,)$ liste

Tableau bidimensionnel

- `A.shape` (n,p) matrice $A \in \mathfrak{M}_{n,p}(\mathbb{K})$
- `A.shape` $(n,1)$ matrice colonne $A \in \mathfrak{M}_{n,1}(\mathbb{K})$

Structure de tableau

Tableau unidimensionnel

- `A.shape` $(n,)$ liste

Tableau bidimensionnel

- `A.shape` (n,p) matrice $A \in \mathfrak{M}_{n,p}(\mathbb{K})$
- `A.shape` $(n,1)$ matrice colonne $A \in \mathfrak{M}_{n,1}(\mathbb{K})$
- `A.shape` $(1,n)$ matrice ligne $A \in \mathfrak{M}_{1,n}(\mathbb{K})$

Type d'un tableau

- Un seul type d'éléments `A.dtype`

Type d'un tableau

- Un seul type d'éléments `A.dtype`
- Par défaut, flottants sur 8 octets.

Type d'un tableau

- Un seul type d'éléments `A.dtype`
- Par défaut, flottants sur 8 octets.
- Par défaut, entiers sur 4 octets :

$$-2^{31} \leqslant x < 2^{31} = 2\,147\,483\,648.$$

Type d'un tableau

- Un seul type d'éléments `A.dtype`
- Par défaut, flottants sur 8 octets.
- Par défaut, entiers sur 4 octets :

$$-2^{31} \leqslant x < 2^{31} = 2\,147\,483\,648.$$

Danger ! Avec les entiers, arithmétique modulo 2^{32}.

Tableaux constants

- `A = np.zeros(3)`

$$\begin{pmatrix} 0. & 0. & 0. \end{pmatrix}$$

`A.shape` $(3,)$

Tableaux constants

- `A = np.zeros(3)`

$$\begin{pmatrix} 0. & 0. & 0. \end{pmatrix}$$

`A.shape` $(3,)$

- `B = np.ones((1,3))`

$$\begin{pmatrix} 1. & 1. & 1. \end{pmatrix}$$

`B.shape` $(3,1)$

Tableaux constants

- `A = np.zeros(3)`

$$\begin{pmatrix} 0. & 0. & 0. \end{pmatrix}$$

`A.shape` $(3,)$

- `B = np.ones((1,3))`

$$\begin{pmatrix} 1. & 1. & 1. \end{pmatrix}$$

`B.shape` $(3,1)$

- `np.ones((3,1))`

$$\begin{pmatrix} 1. \\ 1. \\ 1. \end{pmatrix}$$

Tableaux constants

- `A = np.zeros(3)`

$$\begin{pmatrix} 0. & 0. & 0. \end{pmatrix}$$

`A.shape` $(3,)$

- `B = np.ones((1,3))`

$$\begin{pmatrix} 1. & 1. & 1. \end{pmatrix}$$

`B.shape` $(3,1)$

- `np.ones((3,1))`

$$\begin{pmatrix} 1. \\ 1. \\ 1. \end{pmatrix}$$

- `np.zeros((3,3))`

$$\begin{pmatrix} 0. & 0. & 0. \\ 0. & 0. & 0. \\ 0. & 0. & 0. \end{pmatrix}$$

Tableaux aléatoires

`t` : tuple (format choisi)

- Variables indépendantes de loi uniforme sur $[0,1]$

```
np.random.random(t)
```

Tableaux aléatoires

`t` : tuple (format choisi)

- Variables indépendantes de loi uniforme sur $[0, 1]$

```
np.random.random(t)
```

- Variables indépendantes de loi uniforme sur l'intervalle entier $\{a, \dots, b-1\}$

```
np.random.randint(a, b, t)
```

Tableaux diagonaux

- Matrice identité : `np.identity(d)` I_d

Tableaux diagonaux

- Matrice identité : `np.identity(d)` I_d
- Matrice diagonale : `np.diag(L)` $\mathrm{Diag}(\ell_0, \ldots, \ell_{d-1})$

Autres méthodes

- Progression arithmétique :
  tableau des $x_k = a + k\,\mathrm{d}x$ tels que $a \leqslant x_k < b$

```
np.arange(a, b, dx)
```

Autres méthodes

- Progression arithmétique :
  tableau des $x_k = a + k\,\mathrm{d}x$ tels que $a \leqslant x_k < b$

  `np.arange(a, b, dx)`
- Conversion de liste : `np.array`

Autres méthodes

- Progression arithmétique :
  tableau des $x_k = a + k\,\mathrm{d}x$ tels que $a \leqslant x_k < b$

  `np.arange(a, b, dx)`
- Conversion de liste : `np.array`
- Changements de format

Autres méthodes

- Progression arithmétique :
  tableau des $x_k = a + k\,\mathrm{d}x$ tels que $a \leqslant x_k < b$

  `np.arange(a, b, dx)`
- Conversion de liste : `np.array`
- Changements de format
  - Transposition du tableau bidimensionnel `A` :

    `np.transpose(A)` ou `A.transpose()`

Autres méthodes

- Progression arithmétique :
  tableau des $x_k = a + k\,\mathrm{d}x$ tels que $a \leqslant x_k < b$

  `np.arange(a, b, dx)`
- Conversion de liste : `np.array`
- Changements de format
  - Transposition du tableau bidimensionnel `A` :

    `np.transpose(A)` ou `A.transpose()`
  - Changement de format du tableau `A` :

    `B = A.reshape(f)`

    où le format `f` est compatible avec le format `A.shape`.

Élément d'un tableau

- Tableau unidimensionnel de format $(n,)$:

  `A[i]` pour $0 \leqslant i < n$

Élément d'un tableau

- Tableau unidimensionnel de format $(n,)$:

  `A[i]` pour $0 \leqslant i < n$

- Tableau bidimensionnel de format (n, p) :

  `A[i,j]` pour $0 \leqslant i < n$ et $0 \leqslant j < p$.

Élément d'un tableau

- Tableau unidimensionnel de format $(n,)$:

  `A[i]` pour $0 \leqslant i < n$

- Tableau bidimensionnel de format (n, p) :

  `A[i,j]` pour $0 \leqslant i < n$ et $0 \leqslant j < p$.

- En dimension 3 : `A[i,j,k]`

Tranches d'un tableau bidimensionnel

Tableau `A` de format (n, p)

- i-ème ligne : `A[i,:]` pour $0 \leqslant i < n$

Tranches d'un tableau bidimensionnel

Tableau `A` de format (n, p)

- i-ème ligne : `A[i,:]` pour $0 \leqslant i < n$
- j-ème colonne : `A[:,j]` pour $0 \leqslant j < p$

Tranches d'un tableau bidimensionnel

Tableau `A` de format (n, p)

- i-ème ligne : `A[i,:]` pour $0 \leqslant i < n$
- j-ème colonne : `A[:,j]` pour $0 \leqslant j < p$
- Diagonale : `np.diag(A)`

Tranches d'un tableau bidimensionnel

Tableau `A` de format (n, p)

- i-ème ligne : `A[i,:]` pour $0 \leqslant i < n$
- j-ème colonne : `A[:,j]` pour $0 \leqslant j < p$
- Diagonale : `np.diag(A)`
- Sur-diagonales :
  `np.diag(A, k)` $(a_{i,i+k})_{0 \leqslant i < p-k}$ pour $1 \leqslant k < p$

Tranches d'un tableau bidimensionnel

Tableau `A` de format (n, p)

- i-ème ligne : `A[i,:]` pour $0 \leqslant i < n$
- j-ème colonne : `A[:,j]` pour $0 \leqslant j < p$
- Diagonale : `np.diag(A)`
- Sur-diagonales :
  `np.diag(A, k)` $(a_{i,i+k})_{0 \leqslant i < p-k}$ pour $1 \leqslant k < p$
- Sous-diagonales :
  `np.diag(A, k)` $(a_{j-k,j})_{0 \leqslant j < n+k}$ pour $-n < k \leqslant -1$

Opérations sur les complexes

Tableau `A` de nombres complexes

- Tableau des parties réelles : `A.real()`

Opérations sur les complexes

Tableau `A` de nombres complexes

- Tableau des parties réelles : `A.real()`
- Tableau des parties imaginaires : `A.imag()`

Opérations sur les complexes

Tableau `A` de nombres complexes

- Tableau des parties réelles : `A.real()`
- Tableau des parties imaginaires : `A.imag()`
- Tableau des conjugués : `A.conj()`

Opérations de pivot : Permutation

Sur des lignes : $L_i \leftrightarrow L_j$

```
A[i,:], A[j,:] = A[j,:].copy(), A[i,:].copy()
```

Sur des colonnes : $C_i \leftrightarrow C_j$

```
A[:,i], A[:,j] = A[:,j].copy(), A[:,i].copy()
```

Opérations de pivot : Multiplication

Sur des lignes : $L_i \leftarrow \alpha L_i$

```
A[i,:] = alpha*A[i,:]
```

Sur des colonnes : $C_j \leftarrow \alpha C_j$

```
A[:,j] = alpha*A[:,j]
```

Opérations de pivot : Addition

Sur des lignes : $L_i \leftarrow L_i + \alpha L_j$

```
A[i,:] = A[i,:] + alpha*A[j,:]
```

Sur des colonnes : $C_j \leftarrow C_j + \alpha C_i$

```
A[:,j] = A[:,j] + alpha*A[:,i]
```

Valeurs extrêmes

Pour un tableau `A` de format (n, p)

- Valeurs globales

$$\texttt{A.max()} = \max_{\substack{0 \leqslant i < n \\ 0 \leqslant j < p}} a_{i,j} \qquad \texttt{A.min()} = \min_{\substack{0 \leqslant i < n \\ 0 \leqslant j < p}} a_{i,j}$$

Valeurs extrêmes

Pour un tableau `A` de format (n, p)

- Valeurs globales

$$\texttt{A.max()} = \max_{\substack{0\leqslant i<n\\0\leqslant j<p}} a_{i,j} \qquad \texttt{A.min()} = \min_{\substack{0\leqslant i<n\\0\leqslant j<p}} a_{i,j}$$

- Valeurs par colonnes

$$\texttt{A.max(axis=0)} = \max_{0\leqslant i<n} a_{i,j} \qquad \texttt{A.min(axis=0)} = \min_{0\leqslant i<n} a_{i,j}$$

Valeurs extrêmes

Pour un tableau `A` de format (n, p)

- Valeurs globales

$$\texttt{A.max()} = \max_{\substack{0\leqslant i<n\\0\leqslant j<p}} a_{i,j} \qquad \texttt{A.min()} = \min_{\substack{0\leqslant i<n\\0\leqslant j<p}} a_{i,j}$$

- Valeurs par colonnes

$$\texttt{A.max(axis=0)} = \max_{0\leqslant i<n} a_{i,j} \qquad \texttt{A.min(axis=0)} = \min_{0\leqslant i<n} a_{i,j}$$

- Valeurs par lignes

$$\texttt{A.max(axis=1)} = \max_{0\leqslant j<p} a_{i,j} \qquad \texttt{A.min(axis=1)} = \min_{0\leqslant j<p} a_{i,j}$$

Valeurs extrêmes

Pour un tableau unidimensionnel `A` :

- Plus petit indice i tel que `A[i]` soit maximal : `A.argmax()`
- Plus petit indice i tel que `A[i]` soit minimal : `A.argmin()`

Valeurs extrêmes

Pour un tableau unidimensionnel `A` :

- Plus petit indice i tel que `A[i]` soit maximal : `A.argmax()`
- Plus petit indice i tel que `A[i]` soit minimal : `A.argmin()`

Cas d'un tableau bidimensionnel ?

Calculs de sommes

- Somme de tous les éléments : `A.sum()`

Calculs de sommes

- Somme de tous les éléments : `A.sum()`
- Moyenne de tous les éléments : `A.mean()`

Calculs de sommes

- Somme de tous les éléments : `A.sum()`
- Moyenne de tous les éléments : `A.mean()`
- Calcul des **lois marginales** sachant la loi conjointe d'un couple de variables aléatoires discrètes

Calculs de sommes

- Somme de tous les éléments : `A.sum()`
- Moyenne de tous les éléments : `A.mean()`
- Calcul des **lois marginales** sachant la loi conjointe d'un couple de variables aléatoires discrètes
  - Sommes partielles par colonnes : `A.sum(axis=0)`

$$\sum_{0\leqslant i<n} a_{i,j} = \sum_{0\leqslant i<n} \mathbf{P}(X=i, Y=j) = \mathbf{P}(Y=j)$$

Calculs de sommes

- Somme de tous les éléments : `A.sum()`
- Moyenne de tous les éléments : `A.mean()`
- Calcul des **lois marginales** sachant la loi conjointe d'un couple de variables aléatoires discrètes
  - Sommes partielles par colonnes : `A.sum(axis=0)`

$$\sum_{0\leqslant i<n} a_{i,j} = \sum_{0\leqslant i<n} \mathbf{P}(X=i, Y=j) = \mathbf{P}(Y=j)$$

  - Sommes partielles par lignes : `A.sum(axis=1)`

$$\sum_{0\leqslant j<p} a_{i,j} = \sum_{0\leqslant j<p} \mathbf{P}(X=i, Y=j) = \mathbf{P}(X=i)$$

Calculs de sommes

Si le tableau `A` représente la loi d'une variable aléatoire discrète :

$$a_i = \mathbf{P}(X = i)$$

on peut en déduire la **fonction de répartition** de X :

$$f_i = \mathbf{P}(X \leqslant i) = \sum_{0 \leqslant k \leqslant i} \mathbf{P}(X = k)$$

avec (f_i)=`A.cumsum()`.

Multiplication terme à terme

Deux tableaux `A` et `B` de même format peuvent être multipliés terme à terme.

Multiplication terme à terme

Deux tableaux A et B de même format peuvent être multipliés terme à terme.

Application : **Méthode des rectangles**

- x $= (x_i)_{0 \leqslant i < n}$ abscisses
- y $= \left(f(x_i)\right)_{0 \leqslant i < n}$ ordonnées

Multiplication terme à terme

Deux tableaux `A` et `B` de même format peuvent être multipliés terme à terme.

Application : **Méthode des rectangles**

- `x` $= (x_i)_{0 \leqslant i < n}$ abscisses
- `y` $= \left(f(x_i)\right)_{0 \leqslant i < n}$ ordonnées
- `dx = x[1:] - x[:-1]` $= (x_i - x_{i-1})_{1 \leqslant i < n}$

Multiplication terme à terme

Deux tableaux `A` et `B` de même format peuvent être multipliés terme à terme.

Application : **Méthode des rectangles**

- `x` $= (x_i)_{0\leqslant i<n}$ abscisses
- `y` $= \left(f(x_i)\right)_{0\leqslant i<n}$ ordonnées
- `dx = x[1:] - x[:-1]` $= (x_i - x_{i-1})_{1\leqslant i<n}$
- `(y[1:]*dx).sum()` $= \sum_{1\leqslant i<n} y_i(x_i - x_{i-1}) \approx \int_a^b f(x)\,\mathrm{d}x$
- `(y[:-1]*dx).sum()` $= \sum_{1\leqslant i<n} y_{i-1}(x_i - x_{i-1}) \approx \int_a^b f(x)\,\mathrm{d}x$

Produit matriciel

- Si `A` et `B` sont de formats respectifs (n, p) et (p, q), alors `np.dot(A, B)` donne le produit matriciel $AB \in \mathfrak{M}_{n,q}(\mathbb{K})$.

Produit matriciel

- Si `A` et `B` sont de formats respectifs (n, p) et (p, q), alors `np.dot(A, B)` donne le produit matriciel $AB \in \mathfrak{M}_{n,q}(\mathbb{K})$.
- Si `A` et `x` sont de formats respectifs (n, p) et $(p,)$, alors `np.dot(A, x)` donne la colonne Ax.

Produit matriciel

- Si `A` et `B` sont de formats respectifs (n, p) et (p, q), alors `np.dot(A, B)` donne le produit matriciel $AB \in \mathfrak{M}_{n,q}(\mathbb{K})$.
- Si `A` et `x` sont de formats respectifs (n, p) et $(p,)$, alors `np.dot(A, x)` donne la colonne Ax.
- Si `A` et `x` sont de formats respectifs (n, p) et $(n,)$, alors `np.dot(x, A)` donne la ligne ${}^t xA$.

Fonctions vectorialisées

- Les fonctions du module numpy peuvent s'appliquer aussi bien à un nombre qu'à un tableau (de format quelconque) : si $A = (a_{i,j})_{0 \leqslant i,j < n}$, alors

$$f(A) = \big(f(a_{i,j})\big)_{0 \leqslant i,j < n}.$$

Fonctions vectorialisées

- Les fonctions du module numpy peuvent s'appliquer aussi bien à un nombre qu'à un tableau (de format quelconque) : si $A = (a_{i,j})_{0 \leqslant i,j < n}$, alors

$$f(A) = \left(f(a_{i,j})\right)_{0 \leqslant i,j < n}.$$

- Appliquer une comparaison à un tableau retourne un tableau booléen de même format.

```
A = np.random.random((3, 5))
B, C = (A>.75), (A<=.25)
```

Fonctions vectorialisées

- Les fonctions du module numpy peuvent s'appliquer aussi bien à un nombre qu'à un tableau (de format quelconque) : si $A = (a_{i,j})_{0 \leqslant i,j < n}$, alors

$$f(A) = \big(f(a_{i,j})\big)_{0 \leqslant i,j < n}.$$

- Appliquer une comparaison à un tableau retourne un tableau booléen de même format.

```
A = np.random.random((3, 5))
B, C = (A>.75), (A<=.25)
```

- Un tableau booléen peut être utilisé comme une **fonction indicatrice**, en particulier pour définir des **fonctions en escalier** : `X = 3*B-1*C`.