

Logiciels Scientifiques

Licence Mathématiques

Licence MIASHS

Mineur Mathématiques

2^{ème} année

Cours 11

Andrzej Stos

Tableaux numpy (TP 8 !)

Un *tableau* (ang. *array*) est une structure de données conçue pour faire des calculs vectoriels et matriciels.

Il représente naturellement un tableau de nombres (une matrice).

Exemples :

```
a = np.array([1, 2, 3, 2, 1])  
→ [0 1 2 3 2 1]
```

```
m = np.array([[ 0, 1, 2], [10, 11, 12], [20, 21, 22]])  
→  
[[ 0 1 2]  
 [10 11 12]  
 [20 21 22]]
```

Tableaux vs listes

Tableau ressemble donc à une liste (ou une liste imbriquée), mais se comporte bien différemment !

- ▶ Opérations arithmétiques

```
a = np.array([1, 2, 3, 2, 1])
```

```
b = a + a
```

```
↪ [2 4 6 4 2]
```

Remarque : pour des listes, "+" signifie la concaténation

- ▶ a + 1

```
↪ [2 3 4 3 2]
```

Tableaux vs listes

- ▶ Combiner des tableaux

a.append(0)

↪ **AttributeError: 'numpy.ndarray' object has no attribute 'append'**

b = np.hstack((a, a)) # 1 seul argument !

↪ [1 2 3 2 1 1 2 3 2 1]

l'argument : 1 couple, 1 triplet, 1 quadruplet, ...

b = np.vstack((a, a))

↪ [[1 2 3 2 1]

[1 2 3 2 1]]

Tableaux - forme

```
M = np.array([[1, 2, 3], [4, 5, 6]])
```

```
M.shape
```

```
→ (2, 3)
```

Remarque : *shape* est un *attribut* de l'objet, non pas une fonction
(pas de parenthèses)

```
K = M.T      # transposition
```

```
print(K)
```

```
→ [[1 4]
```

```
    [2 5]
```

```
    [3 6]]
```

```
A = np.array([1, 2, 3, 4, 5])
```

```
A.shape
```

```
→ (5,)
```

tableau 1-dim, de longueur 5, *sans forme algébrique*.

Il peut représenter un *vecteur ligne* ou bien un *vecteur colonne*.

Tableaux - construction

Rappel :

- ▶ `np.arange(a, b, pas)` : des points équidistants dans $[a, b[$
- ▶ `np.linspace(a, b, n)` : n points équidistants dans $[a, b]$
- ▶ `np.random.rand(n)` : n nombres aléatoires dans $[0, 1]$
- ▶ `np.random.randint(a, b, n)` : n entiers aléatoires dans $[a, b[$

Nouveaux :

- ▶ `np.zeros((n, k))` : tableau de taille $n \times k$ rempli de 0 (de type float par défaut)
- ▶ `np.zeros((n, k), dtype=int)` : de même, du type int
- ▶ `np.ones((n, k))` : tableau de taille $n \times k$ rempli de 1.0
- ▶ `np.diag(v)` où v est un vecteur : matrice ayant v sur la diagonale, 0 ailleurs. Exemple : `diag([1, 2, 3])` \hookrightarrow
[[1 0 0]
 [0 2 0]
 [0 0 3]]

Tableaux - accès aux éléments

Soit M le tableau suivant :

```
[[ 0  1  2]
 [10 11 12]
 [20 21 22]]
```

- ▶ $M[0, 0] \hookrightarrow 0$
- ▶ $M[1, :] \hookrightarrow [10 11 12]$ (tableau "sans forme")
- ▶ $M[:, 2] \hookrightarrow [2 12 22]$
- ▶ $M[::2, :] \hookrightarrow$ toutes les 2 lignes, toutes les colonnes :

```
[[ 0  1  2]
 [20 21 22]]
```

Tableaux - règles de calcul

Un tableau numpy *n'est pas* un vecteur ou une matrice d'algèbre linéaire : **les règles de calcul sont bien différentes !**

En principe, calcul se fait **composante par composante** :

```
A = np.array([[1, 2], [3, 4]])
```

```
↪ [[1 2]  
     [3 4]]
```

```
print(A * A)
```

```
↪ [[1 4]  
     [9 16]]
```

Tableaux - règles de calcul

Extension automatique (ang. *broadcasting*)

```
x = np.array([1, 2, 3])
```

```
x + 10
```

```
↪ [11, 12, 13]
```

Extension :

10 se comporte comme [10, 10, 10] :

$$[1, 2, 3] + [10, 10, 10] = [11, 12, 13]$$

Tableaux - règles de calcul

Extension automatique (ang. *broadcasting*)

A = np.array([[1, 2], [3, 4], [5, 6]])

↪ [[1, 2]
[3, 4]
[5, 6]]

B = np.array([2, 3])

A * B = ??

B est *étendu verticalement* pour égaler la taille de A :

$$\begin{array}{ccc} [[1, 2] & [[2, 3] & [[2, 6] \\ [3, 4] & * & [2, 3] & = & [6, 12] \\ [5, 6]] & [2, 3]] & [10, 18]] \end{array}$$

Extension verticale : le nombre d'éléments de B doit être le même que le nombre de colonnes de A.

NB. Ce produit est *commutatif* : A * B vaut B * A.

Tableaux - quelques pièges

Soit $A = \text{np.array}([1, 2, 3])$

- ▶ Affectation $B = A$ **ne crée pas** un nouveau tableau

Exemple (déjà vu avec les listes) :

$B[0] = 10$

`print(A)`

↪ [10 2 3]

- ▶ $B = A.reshape(n, k)$ **ne crée pas** un nouveau tableau (mais un nouveau *regard* sur le même endroit dans la mémoire).
- ▶ La transposition $B = A.T$ ou $B = \text{np.transpose}(A)$ **ne crée pas** un nouveau tableau (mais un nouveau *regard*).

Créer une copie *indépendante* d'une matrice (nouveau endroit dans la mémoire) :

$B = \text{np.copy}(A)$

Tableaux - tests, comparaisons

Résultat d'une comparaison de deux tableaux est un tableau :

```
A = np.array([1, 2, 3])
B = np.array([1, 5, 6])
print(A == B)
↪ [True False False]
```

Son utilisation dans un if est *ambiguë* !

```
if A == B:
↪ erreur...
```

Pour être précis :

```
if np.all(A == B):  # vrai si toutes composantes égales
if np.any(A == B):  # vrai si au moins une composante égales
```

Tableaux - sélection

```
Soit A = np.array([1, 2, 3])  
print(A > 1)  
↪ [False True True]
```

On peut l'utiliser pour sélection :

```
print(A[A > 1])  
↪ [2 3]
```

Exemple : trouver les noms des communes françaises de plus de 100000 habitants :

```
noms[nom > 100000]  
(ne fonctionne pas avec les listes !)
```

Les visualiser sur une carte de France :

```
x = lon[nom > 100000]  
y = lat[nom > 100000]  
plt.plot(x, y, '.')
```

Tableaux - algèbre linéaire : np.linalg

Soit A, B deux tableaux.

- ▶ **Produit matriciel** : dot

`y = A.dot(B)` # si les tailles sont cohérentes !!

- ▶ Matrice inverse : `A1 = np.linalg.inv(A)`

- ▶ Déterminant `np.linalg.det(A)`

- ▶ Résolution d'un système d'équations linaires $Ax = b$:

- ▶ l'inversion de la matrice (`inv`)

- ▶ `np.linalg.solve(A, b)` (solution numérique, non symbolique !)

- ▶ Valeurs propres, vecteurs propres :

`val, vect = np.linalg.eig(A)`

- ▶ val est un tableau des valeurs propres ; les valeurs propres multiples sont répétées.

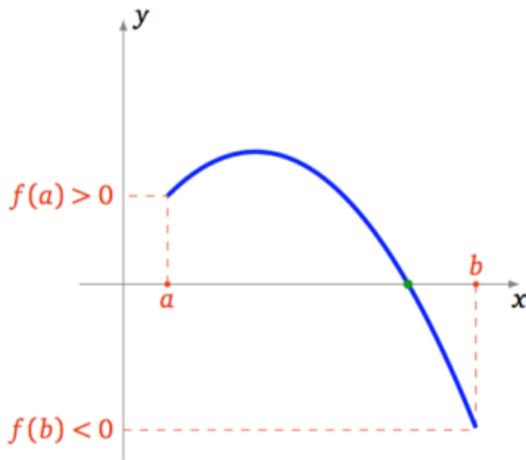
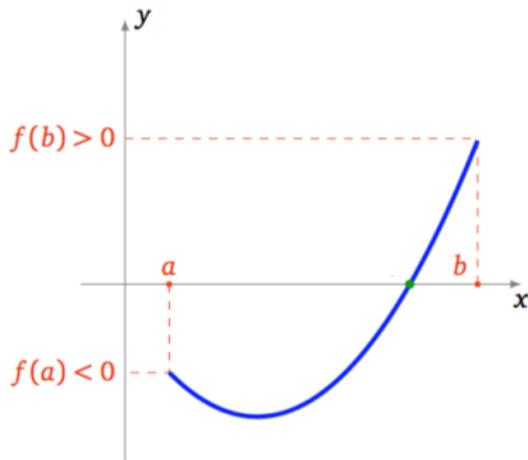
- ▶ vect est un tableau bi-dimensionnel (matrice) composé des vecteurs propres :

La i -ème colonne `vect[:, i]` donne le vecteur propre correspondant à `val[i]`.

La norme euclidienne de chaque vecteur vaut 1.

Méthodes numériques - dichotomie (TP 8)

Soit $I = [a, b]$ un intervalle et f une fonction continue sur I .
Supposons que $f(a)f(b) < 0$.



Crédit image : Exo7

Alors il existe (au moins un) $x \in [a, b]$ tel que $f(x) = 0$.
Objectif : trouver un $x \in I$ à tel que $|f(x)| < \varepsilon$.

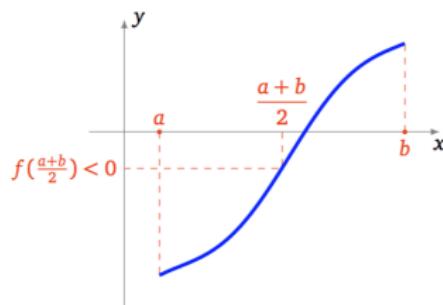
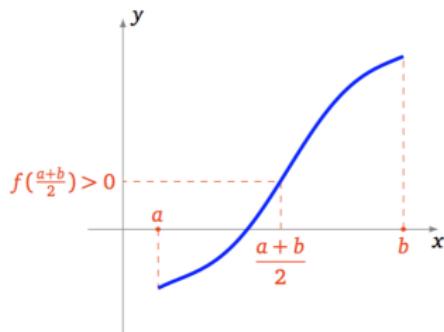
Méthodes numériques - dichotomie (TP 8)

Dichotomie :

- ▶ Tant que $|f(\text{milieu de } I)| > \varepsilon$:

Notons c le milieu de I .

Si $f(a)f(c) < 0$, alors $I = [a, c]$
sinon $I = [c, b]$



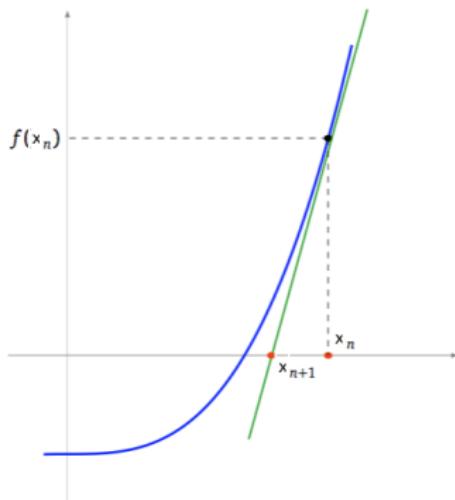
Crédit image : Exo7

- ▶ A la fin, retourner c .

Remarque : À chaque répétition l'intervalle I est deux fois plus court. Critère d'arrêt alternatif : tant que $|b - a| > \varepsilon$.

Méthodes numériques - Newton (TP 8)

Soit f une fonction dérivable sur un intervalle fermé I qui possède une racine simple $r \in I$ (telle que $f(r) = 0, f'(r) \neq 0$).



Soit $x_0 \in I$ suffisamment proche de r .

Tangente en x_0 :

$$y = f'(x_0)(x - x_0) + f(x_0)$$

Intersection avec l'axe des abscisses :

$$0 = f'(x_0)(x - x_0) + f(x_0)$$

$$\text{Solution : } x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Formule générale :

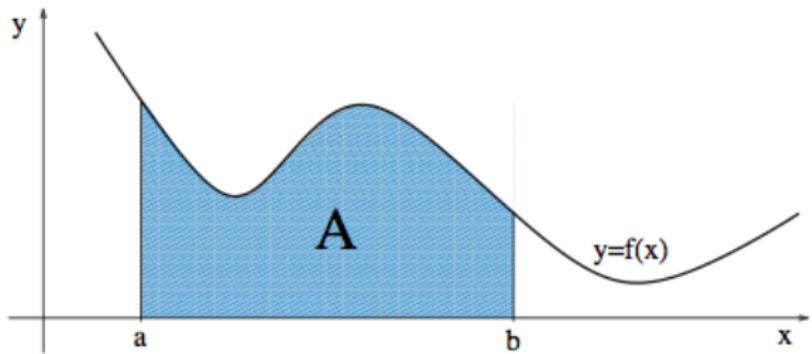
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Critère d'arrêt : tant que $|f(x_n)| > \varepsilon$

Crédit image : Exo7

Méthodes numériques - intégration (TP 8)

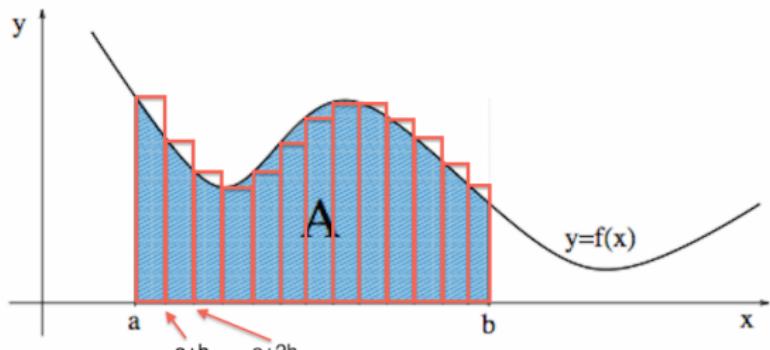
$$A = \int_a^b f(x)dx$$



Méthodes numériques - intégration (TP 8)

$$A = \int_a^b f(x) dx$$

Méthode
des rectangles



$n + 1$ points équirépartis dans $[a, b]$ (donc n rectangles) :

$$a = x_0 < \dots < x_{n-1} < x_n = b$$

Python : `x = np.linspace(a, b, n+1)`

Posons $h = (b - a)/n$ (largeur d'un rectangle) et $y_i = f(x_i)$

$$S_g = h \sum_{i=0}^{n-1} y_i$$

$$S_d = h \sum_{i=0}^{n-1} y_{i+1} = h \sum_{i=1}^n y_i$$

Rectangles à gauche

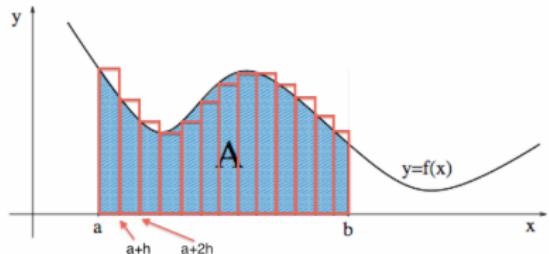
Rectangles à droite

Méthodes numériques - intégration (TP 8)

Méthode des trapèzes

Observation :

l'aire de trapèze rectangle
= la moyenne des aires de deux rectangles.



$$S_t = h \sum_{i=0}^{n-1} \frac{y_i + y_{i+1}}{2}$$

En simplifiant la somme télescopique :

$$S_t = h \left(\frac{y_0 + y_n}{2} + \sum_{i=1}^{n-1} y_i \right)$$