INTRODUCTOIN A PYTHON

Drs KYELEM YACOUBA

Université Joseph Ki-Zerbo

Laboratoire de Mathématiques et d'Informatique (LAMI)

Contact : kyelemy@yahoo.fr/71025937

Plan du cours

Chapitre 4 : Les tableaux en python

I. Définition

Un tableau est un type de donnée permettant de stocker plusieurs valeurs de manière séquentielle et d'y accéder à l'aide d'une seule variable. un tableau est une collection d'éléments du même type.

Nous allons voir comment créer des tableaux avec la fonction numpy.array() de NumPy. Ces tableaux pourront être utilisés comme des vecteurs ou des matrices grâce à des fonctions de NumPy (numpy.dot(), numpy.linalg.det(), numpy.linalg.inv(), numpy.linalg.eig(), etc.) qui permettent de réaliser des calculs matriciels utilisés en algèbre.

Nous devons importer le module numpy. Pour cela, il suffit de faire : import numpy as np.

Pour créer des tableaux, nous allons utiliser numpy.array().

II. Tableau unidimensionnel

Pour créer un tableau **unidimensionnel**, il suffit de passer une liste de nombres en argument de numpy.array(). Une liste est constituée de nombres séparés par des virgules et entourés de crochets ([et]).

```
>>>a = np.array([4,7,9])
>>> a
array([4, 7, 9])
```

Pour connaître le type du résultat de numpy.array(), on peut utiliser la fonction type().
>>>type(a)

On constate que ce type est issu du package numpy. Ce type est différent de celui d'une liste.

```
>>>type([4, 7, 9])
list
```

III. Tableau multidimensionnel

Pour créer un tableau bidimensionnel, il faut transmettre à numpy.array() une liste de listes grâce à des crochets imbriqués.

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

```
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
```

La fonction numpy.size() renvoie le nombre d'éléments du tableau. La fonction numpy.shape() renvoie la taille du tableau.

IV. Opérations sur les tableaux

1. *Produit terme à terme*

Il est possible de réaliser un produit terme à terme grâce à l'opérateur *. Il faut dans ce cas que les deux tableaux aient la même taille.

```
>>>a = np.array([[1, 2, 3],
                  [4, 5, 6]])
>>> b = np.array([[2, 1, 3],
                  [3, 2, 1]])
>>> a*b
array([[ 2,  2,  9],
       [12, 10,  6]])
```

2. *Produit matriciel*

Un tableau peut jouer le rôle d'une matrice si on lui applique une opération de calcul matriciel. Par exemple, la fonction numpy.dot() permet de réaliser le produit matriciel.

```
>>>a = np.array([[1, 2, 3],
                  [4, 5, 6]])
>>> b = np.array([[4],
                  [2],
                  [1]])
>>> np.dot(a,b)
array([[11],
       [32]])
```

Remarque : on peut utiliser les opérateurs +,- et / pour faire des opérations en python.

V. Quelques fonctions de génération de tableau

Tableaux de 0 - numpy.zeros()

zeros(n) renvoie un tableau 1D de n zéros.

```
>>>np.zeros(3)
```

```
array([ 0.,  0.,  0.])
```

zeros((m,n)) renvoie tableau 2D de taille m x n, c'est-à-dire de shape (m,n).

```
>>>np.zeros((2,3))
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

Tableaux de 1 - numpy.ones()

ones(n) renvoie un tableau 1D de n uns. ones((m,n)) renvoie tableau 2D de taille m x n.

Matrice identité - numpy.eye()

eye(n) renvoie tableau 2D carré de taille n x n, avec des uns sur la diagonale et des zéros partout ailleurs.

```
>>>np.eye(3)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

VI. Tracé de courbes

Pour tracer des courbes, Python n'est pas suffisant et nous avons besoin de la bibliothèque Matplotlib.

nous présentons deux styles de programmation :

- ✓ le style « pyplot » qui utilise directement des fonctions du module pyplot ;
- ✓ le style « Orienté Objet (OO) » qui est recommandé dans la documentation de Matplotlib.

Quel que soit le style de programmation, il est nécessaire d'importer le module pyplot de matplotlib. On lui donne en général l'alias plt. On précise ensuite ce module lors de l'appel des fonctions.

VII. Création d'une courbe

Utilisation de plot()

L'instruction plot() permet de tracer des courbes qui relient des points dont les abscisses et ordonnées sont fournies en arguments.

Style pyplot

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

plt.plot([1, 3, 4, 6], [2, 3, 5, 1])

plt.show()
```

Style « Orienté Objet »

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

fig, ax = plt.subplots()
ax.plot([1, 3, 4, 6], [2, 3, 5, 1])

plt.show()
```

Nous allons tracer la fonction cosinus.

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 30)
y = np.cos(x)
fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```

1. *Définition du domaine des axes - xlim(), ylim() et axis()*

Dans le style pyplot, il est possible fixer les domaines des abscisses et des ordonnées en utilisant les fonctions xlim(), ylim() ou axis().

xlim(xmin, xmax)

ylim(ymin, ymax)

axis() permet de fixer simultanément les domaines des abscisses et des ordonnées. axis([xmin, xmax, ymin, ymax]). Dans le style « Orienté Objet », il faut utiliser set_xlim().

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 30)
y = np.cos(x)
plt.plot(x, y)
```

```
plt.xlim(-1, 5)
plt.show()
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 30)
y = np.cos(x)
fig, ax = plt.subplots()
ax.plot(x, y)
ax.axis([-1, 5, -2, 2])
plt.show()
```

2. *Ajout d'un titre - title()*

Dans le style pyplot, on peut ajouter un titre grâce à l'instruction title().

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 30)
y = np.cos(x)
plt.plot(x, y)
plt.title("Fonction cosinus")
plt.show()
```

Dans le style « Orienté Objet », il faut utiliser set_title().

3. *Ajout d'une légende - legend()*

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 30)
y = np.cos(x)
fig, ax = plt.subplots()
ax.plot(x, y, label="cos(x)")
ax.legend()
plt.show()
```

4. *Labels sur les axes*

Dans le style pyplot, des labels sur les axes peuvent être ajoutés avec les fonctions xlabel() et ylabel().

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 30)
y = np.cos(x)
plt.plot(x, y)
plt.xlabel("abscisses")
```

```
plt.ylabel("ordonnées")
plt.show()
```

Dans le style « Orienté Objet », il faut utiliser set_xlabel() et set_ylabel().

5. *Affichage de plusieurs courbes*

Pour afficher plusieurs courbes sur un même graphe, on peut procéder de la façon suivante :

Style « pyplot »

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 30)
y1 = np.cos(x)
y2 = np.sin(x)
plt.plot(x, y1)
plt.plot(x, y2)
plt.show()
```

Style « Orienté Objet »

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 30)
y1 = np.cos(x)
y2 = np.sin(x)
fig, ax = plt.subplots()
ax.plot(x, y1)
ax.plot(x, y2)
plt.show()
```

References

https://courspython.com/tableaux-numpy.html

https://courspython.com/introduction-courbes.htmhttps://waytolearnx.com/2020/06/les-tableaux-en-python.html

https://www.ukonline.be/cours/numcomp/decouvrir-scipy/chapitre2-2l

Cours Matthieu Boileau & Vincent Legoll Apprendre Python pour les sciences CNRS - Université de Strasbourg.

Cours de Python Introduction à la programmation Python pour la biologie Patrick Fuchs et Pierre Poulain https://python.sdv.univ-paris-diderot.fr/ .