Python pour les data scientistes

Chapitre 4 : Le package Numpy

1. Introduction

La bibliothèque NumPy (http://www.numpy.org/) permet d'effectuer des calculs numériques sur des vecteurs ou des matrices, élément par élément, via un nouveau type d'objet appelé array.

NumPy est un outil performant pour la manipulation de tableaux à plusieurs dimensions. Le type **array** est similaire à une liste, mais dont tous les éléments sont du même type : des entiers, des flottants ou des booléens.

Le module NumPy possède des fonctions basiques en algèbre linéaire, ainsi que pour les transformées de Fourier.

Dans ce chapitre, nous nous focaliserons sur les outils les plus importants de la librairie NumPy.

Pour commencer, il faut installer la librairie NumPy (sauf pour l'environnement anaconda) :
Tapez à l'invité : **pip install numpy**

Maintenant, il faut charger le module NumPy :

import numpy

On peut également définir un nom raccourci pour NumPy :
import numpy as np

Ceci permettra d'utiliser la variable np à la place de numpy pour appeler les variables et méthodes de la librairie après toutes les fonctions NumPy seront alors préfixées par np.

Par exemple, NumPy permet d'obtenir la valeur de pi :

```
>>> import numpy as np
>>> np.pi
3.141592653589793
```

2. Introduction aux tableaux

Python offre différentes possibilités pour manipuler des tableaux de différentes formes.

- Pour créer un tableau :

Les tableaux peuvent être créés avec la commande **numpy.array()**. On utilise des crochets pour délimiter les listes d'éléments dans les tableaux.

```
import numpy as np
tableau = np.array([1, 2, 3])
tableau
```

Ici on a créé un tableau de trois éléments et on l'a affiché.

On vérifie maintenant le type du tableau avec : type(a)

numpy.ndarray

On voit que l'on a obtenu un objet de type **numpy.ndarray**.

Par défaut les éléments d'un tableau sont des float (un réel en double précision) ; mais on peut donner un deuxième argument qui précise le type (int, complex, bool, ...).

Par exemple :
a=np.array([1,2], dtype=int)
a

- Accéder à un élément du tableau :

Comme pour les listes, et les chaines de caractères les indices des éléments commencent à zéro. Pour les matrices (ou tableaux de dimension 2), le premier index se réfère à la ligne, le deuxième à la colonne.

Pour accéder à un élément du tableau, on utilise l'identificateur du tableau avec l'indice de l'élément, par exemple : a[2] accéder au 3ème élément du tableau.

- Quelques commandes usuelles :

- Remplir un tableau avec des zéros : **np.zeros(i)** où i représente le nombre d'éléments du tableau
- Matrice identité d'ordre i: np.identity(i).
- np.empty : tableau vide.
- np.ones(i) est le vecteur [1 1 1 1 1 1……1] avec i éléments.
- np.eye(i,j) est la matrice à i lignes et j colonnes contenant des 1 sur la diagonale et des zéros partout ailleurs.
- a.shape : retourne les dimensions du tableau a.
- a.dtype : retourne le type des éléments du tableau a.
- a.size : retourne le nombre total d'éléments du tableau a
- a.ndim : retourne la dimension du tableau (1 pour un vecteur, 2 pour une matrice)
- Transposée d'une matrice a : a.T

- Il est possible de créer un tableau à deux dimensions en utilisant une liste de listes au moyen de crochets imbriqués. Les listes internes correspondent à des lignes du tableau. Pour cela, il suffit de passer en argument une liste de listes à la fonction array() :

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

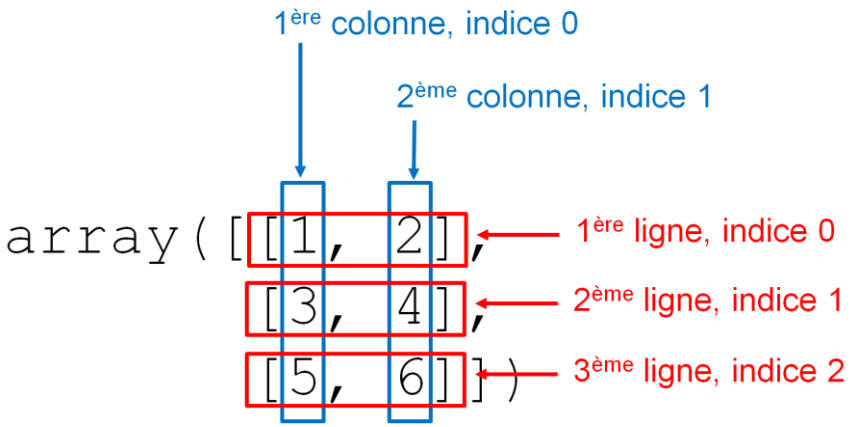


tableau = np. array ([[1 , 2], [3, 4], [5, 6]])
tableau
array ([[1 , 2],
[3, 4],
[5, 6]])

- Pour créer un tableau multidimensionnel :

On peut aussi créer des tableaux à trois dimensions en passant comme argument à la fonction array() une liste de listes :
x = np. array ([[[1 , 2], [2, 3]] , [[4 , 5], [5, 6]]])
x
array ([[[1 , 2],
[2, 3]] ,
[[4 , 5],
[5, 6]]])

La fonction array() peut créer des tableaux à n'importe quel nombre de dimensions. Toutefois ça devient vite compliqué lorsqu'on dépasse trois dimensions. Retenez qu'un objet array à une dimension peut être considéré comme un vecteur et un array à deux dimensions comme une matrice.

- numpy.arange() et range() :

numpy.arange() retourne un objet de type numpy.ndarray.
range() retourne un objet de type range.

m = np.arange(3, 15, 2)
m
array([3, 5, 7, 9, 11, 13])
type(m)
numpy.ndarray

n = range(3, 15, 2)
n
range(3, 15, 2)
type(n)
range

- Création d'un tableau avec une séquence de nombre

La fonction **linspace(premier, dernier, n)** renvoie un tableau unidimensionnel commençant par premier, se terminant par dernier avec n éléments régulièrement espacés.

Soit l'exemple suivant qui consiste à créer un tableau avec 9 éléments de la suite des nombres -4, -3, …., 4 :

```
a = np.linspace(-4, 4, 9)
a
array([-4., -3., -2., -1., 0., 1., 2., 3., 4.])
```

- Copie d'un tableau

Un tableau est un objet. Si l'on affecte un tableau A à un autre tableau B, A et B font référence au même objet. En modifiant l'un on modifie donc automatiquement l'autre. Pour éviter cela on peut faire une copie du tableau A dans le tableau B moyennant la fonction copy(). Ainsi, A et B seront deux tableaux identiques, mais ils ne feront pas référence au même objet.

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

3. 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]])
```

Le produit d'une matrice de taille n x m par une matrice m x p donne une matrice n x p.
A partir de la version 3.5 de Python, il est également possible d'effectuer le produit matriciel en utilisant @.

```
a @ b
array([[11],
       [32]])
```

Attention, si le deuxième argument est un vecteur ligne, il sera transformé si besoin en vecteur colonne, par transposition. Ceci est dû au fait qu'il est plus simple de définir un vecteur ligne,
par exemple v=np.array([0,1,2]), qu'un vecteur colonne, ici v=np.array([[0],[1],[2]]).

Voilà un exemple :

```
a=np.array([[1,1,1],[0,1,1],[0,0,1]]
v=np.array([0,1,2])
np.dot(v,A)
array([0, 1, 3])

np.dot(A,v)
array([3, 3, 2])
```

On peut aussi transposer préalablement un vecteur ligne :

```
v=np.array([[0,1,2]])
```

v=np.transpose(v)
v

L'opération A**2 correspond à une élévation au carré terme à terme. Pour lever une matrice au carré il faut taper np.dot(A,A).

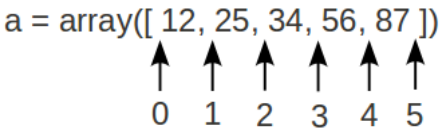
Pour lever une matrice carrée à une puissance n il faut faire une boucle :
B=A.copy()
for i in range(1,n):
... B = np.dot(A,B)
A la sortie de cette boucle B contiendra An.

4. Extraire un sous tableau

Lors de la manipulation des tableaux, on a souvent besoin de récupérer une partie d'un tableau. Pour cela, Python permet d'extraire des *tranches* d'un tableau grâce une technique appelée **slicing** (tranchage, en français). Elle consiste à indiquer entre crochets des indices pour définir le début et la fin de la *tranche* et à les séparer par deux-points :.

np.array([12, 25, 34, 56, 87])
a[1:3]
array([25, 34])

Dans la tranche [n:m], l'élément d'indice n est inclus, mais pas celui d'indice m. Un moyen pour mémoriser ce mécanisme consiste à considérer que les limites de la tranche sont définies par les numéros des positions situées entre les éléments, comme dans le schéma ci-dessous :



Il est aussi possible de ne pas mettre de début ou de fin.

a[1:]
array([25, 34, 56, 87])
a[:3]
array([12, 25, 34])
a[:]
array([12, 25, 34, 56, 87])

5. Fonctions mathématiques de NumPy

NumPy dispose d'un grand nombre de fonctions mathématiques qui peuvent être appliquées directement à un tableau. Dans ce cas, la fonction est appliquée à chacun des éléments du tableau.

- Fonctions trigonométriques

numpy.sin(x)	sinus
numpy.cos(x)	cosinus
numpy.tan(x)	tangente
numpy.arcsin(x)	arcsinus
numpy.arccos(x)	arccosinus
numpy.arctan(x)	arctangente

– Fonctions diverses

x**n	x à la puissance n, exemple : x**2
numpy.sqrt(x)	racine carrée
numpy.exp(x)	exponentielle
numpy.log(x)	logarithme népérien
numpy.abs(x)	valeur absolue
numpy.sign(x)	signe

– Arrondis

numpy.around(x,n)	arrondi à n décimales
numpy.trunc(x)	retourne la partie entière du nombre (le nombre est tronqué)

np.array([3.73637, 5.4374345])
np.around(x,2)
array([3.74, 5.44])

around(x,0) retourne l'entier le plus proche.

np.around(x,0)
array([4., 5.])