La simulation numérique

Numpy et Matplotlib

Les modules incontournables du calcul scientifique

Introduction

Ce document représente un rappel du module Numpy du Python. Il a pour but de présenter l'essentiel grâce à des exemples à saisir dans la console (sous Idle).

Le module Numpy : Présentation

Le calcul scientifique nécessite la manipulation de données en quantité souvent importante.

NumPy (Numerical Python) est une bibliothèque Python pour travailler avec les grands ensembles de données de manière efficace. Le module numpy est la boîte à outils indispensable pour faire du calcul scientifique avec Python.

Pour modéliser les vecteurs, les matrices, et plus généralement les tableaux à n dimensions, numpy fournit le type ndarray (N dimensional array).

Il y a des différences majeures avec les listes (resp. les listes de listes) qui pourraient elles aussi nous servir à représenter des vecteurs (resp. des matrices) :

- Les tableaux numpy sont homogènes, c'est-à-dire constitués d'éléments du même type. On trouvera donc des tableaux d'entiers, des tableaux de flottants, etc.
- La taille des tableaux numpy est fixée à la création. On ne peut donc augmenter ou diminuer la taille d'un tableau comme le ferait pour une liste (à moins de créer un tout nouveau tableau, bien sûr).

Ce module n'est pas fourni avec la distribution Python de base, il doit être installé à partir du site officiel : https://numpy.org/install/

Le module numpy (https://numpy.org/doc/stable/reference/arrays.ndarray.html) propose plusieurs sous-modules intéressants, par exemple :

numpy.linalg : un module d'algèbre linéaire basique,

numpy.random : un module pour la génération des nompbres aléatoires,

Comment charger le module numpy ?

```
>>> import numpy as np
```

Remplir un tableau numpy

`a = np.array( [1, 2, 3] )`	`array([1, 2, 3])`
`b = a.tolist()`	`[1, 2, 3]`
`c = np.arange (3,10,2,dtype=int)`	`array([3, 5, 7, 9])`
`d = np.array([2]*5)`	`array([2, 2, 2, 2, 2])`
`e = np.linspace (2, 5, n=3)` `Par défaut: n = 50`	`array([ 2. ,  3.5,  5. ])`

`f = np.ones((3, 5));`	`Array([[ 1.,  1.,  1.,  1.,  1.],`
	`[ 1.,  1.,  1.,  1.,  1.],`
	`[ 1.,  1.,  1.,  1.,  1.]])`

`g = np.zeros((3, 5));`	`array([[ 0.,  0.,  0.,  0.,  0.],`
	`[ 0.,  0.,  0.,  0.,  0.],`
	`[ 0.,  0.,  0.,  0.,  0.]])`

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

`np.diag([1,2,3]);`	`array([[1, 0, 0],`
	`[0, 2, 0],`
	`[0, 0, 3]])`

`np.diag([1,2,3],1);`	`array([[0, 1, 0, 0],`
	`[0, 0, 2, 0],`
	`[0, 0, 0, 3],`
	`[0, 0, 0, 0]])`

`np.concatenate((f,g),axis=0);`
Premet de concaténer (joindre) deux tableaux le long d'un axe spécifié.
 	
En NumPy, l'argument **axis** est utilisé pour spécifier l'axe le long duquel une opération doit être effectuée.
axis=0 correspond à l'axe des lignes (empiler les lignes),
L'axe 0 pointe vers le bas (verticalement).
	`array([[ 1.,  1.,  1.,  1.,  1.],`
	`[ 1.,  1.,  1.,  1.,  1.],`
	`[ 1.,  1.,  1.,  1.,  1.],`
	`[ 0.,  0.,  0.,  0.,  0.],`
	`[ 0.,  0.,  0.,  0.,  0.],`
	`[ 0.,  0.,  0.,  0.,  0.]`
	`])`

`np.concatenate((f,g),axis=1);`
 	
axis=1 correspond à l'axe des colonnes,
L'axe 1 pointe vers la droite (horizontalement).
 	
Les éléments de g seront ajoutés à la droite de chaque ligne de f)
	`Array([[ 1.,1.,1.,1.,1., 0.,0.,0.,0.,0.],`
	`[ 1.,1.,1.,1.,1., 0.,0.,0.,0.,0.],`
	`[ 1.,1.,1.,1.,1., 0.,0.,0.,0.,0.]`
	`])`

`np.column_stack((a,a,a));`
Permet d' empiler les tableaux en colonnes.
	`array([[1, 1, 1],`
	`[2, 2, 2],`
	`[3, 3, 3]])`

`a.fill(3.14);`	`array([3, 3, 3])`

`def f(i,j):`
`return 10*i+j`
`t1 = np.fromfunction(f,(4,5));`
`print(t1)`
Equivalent à:
`t2 = np.array([[f(i,j) for j in range(5)] for i in range(4)]);`
`print(t2)`
	Permet de construire un tableau dont le terme général obéit à une formule donnée.
	Résultat:
	`array([[ 0,  1,  2,  3,  4],`
	`[10, 11, 12, 13, 14],`
	`[20, 21, 22, 23, 24],`
	`[30, 31, 32, 33, 34]])`

`import random` `random.randint(1,5);`	`2`

Taille des tableaux

La dimension d'une matrice:	`f.ndim`	→	`2`
Le nombre d'éléments d'une matrice:	`f.size`	→	`15`
Le nombre de lignes d'une matrice:	`np.size(f,0)`	→	`3`
Le nombre de colonnes d'une matrice:	`np.size(f,1)`	→	`5`
La taille/cardinalité d'une matrice	`f.shape`	→	`(3, 5)`

Modifier la Taille d'un tableau

```
f.reshape(1,15)
f.shape = (1,15)
f.shape = 15
```

```
f = array([[ 1.,  1.,  1.,  1.,  1.],
           [ 1.,  1.,  1.,  1.,  1.],
           [ 1.,  1.,  1.,  1.,  1.]])
Résultat: array([[1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.,1.]])
```

Supprimer, insérer, ajouter des élements/lignes/colonnes :
axe des lignes(axis=0); axe des colonnes(axis=1)

Supprimer l'élément ou la ligne d'indice 2	np.delete(t,2,axis=0);	array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])	array([[1,2,3], [4,5,6]])
Supprimer l'élément ou la colonne d'indice 2	np.delete(t,2, axis=1);	array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])	array([[1, 2], [4, 5], [7, 8]])
Supprimer les colonnes d'indices 0 et 2	np.delete(t,[0,2],axis=1)	array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])	array([[2], [5], [8]])
Insérer des -1 juste avant la colonne d'indice 2	np.insert(t,2,-1,axis=1)	array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])	array([[1,2,-1,3], [4,5,-1,6], [7,8,-1,9]])
Insérer des -1 juste avant la ligne d'indice 2	np.insert(t,2,-1,axis=0)	array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])	array([[1, 2, 3], [4, 5, 6], [-1, -1, -1], [7, 8, 9]])
Ajouter une colonne après la dernière colonne de la matrice	np.append(t, **[**[10],[11],[12]**]**, axis=1)	array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])	array([[1, 2, 3, 10], [4, 5, 6, 11], [7, 8, 9, 12]])
Ajouter une ligne après la dernière ligne de la matrice	np.append(t,**[**[10,11,12]**]**, axis=0)	array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])	array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])

Lecture dans un tableau à 1 dimension
V [indice_début(inclu) : indice_fin(exclu) : incrément]

V[0] ⇔ v[-v.size]	Premier élément
v[-1] ⇔ v[v.size-1]	Dernier élément
v[4:11]	Les éléments d'indice 4 à 10
v[4:11:2]	Les éléments d'indice 4 à 10 de pas = 2
v[:]	Tous les éléments
v[::-1]	Tous les éléments dans le sens inverse
v[[6,2,1,3]] ⇔ np.take(v, [6,2,1,3])	Les éléments d'indice 6, 2, 1, et 3

Lecture dans un tableau à 2 dimensions
M[numero_ligne : numero_colonne]
M[ligne_début(inclue):ligne_fin(exclue), col_début(inclue):col_fin(exclue)]

Si M est une matrice d'ordre (n , p) :

M[a,b]	L'élément en position (a,b)
M[a] ⇔ M[a,:]	Vecteur ligne en position a

```
M[:,b]          Vecteur colonne en position b
M[a:b,c:d]      Sous-matrice de lignes de a à b-1 et de colonnes de c à d-1
M[:,:]          Une copie intégrale de M avec nouvelle référence
                Retourne une sous-matrice:
M[:a,:b]          - Les lignes du début jusqu'à a exclu.
                  - Les colonnes de début jusqu'à b exclu.
M[::-1]         On inverse l'ordre des lignes
M[:,::-1]       On inverse l'ordre des colonnes
M[::2,::2]      Lignes et colonnes paires
M[1::2,1::2]    Lignes et colonnes impaires
```

Fonctions/opérateurs vectorisées

Soient les deux matrices suivantes:

```
a = array([[1, 2, 3],          b = array([[1, 2, 3],
           [4, 5, 6],                     [4, 5, 6],
           [7, 8, 9]])                    [7, 8, 9]])
```

`np.negative(a) ⇔ -a`	`array([[-1, -2, -3],`
	`[-4, -5, -6],`
	`[-7, -8, -9]])`

`print(a + b) ⇔ np.add(a, b)`	`array([[ 2, 4, 6],`
	`[ 8, 10, 12],`
	`[14, 16, 18]])`

`print(a - b ⇔ np.subtract(a, b)`	`array([[0, 0, 0],`
	`[0, 0, 0],`
	`[0, 0, 0]])`

`print(a / b) ⇔ np.divide(a,b)`	`array([[ 1., 1., 1.],`
	`[ 1., 1., 1.],`
	`[ 1., 1., 1.]])`

`print(a ** b) ⇔ np.power(a,b)`	`array([[ 1, 4, 27],`
	`[ 256, 3125, 46656],`
	`[ 823543, 16777216, 387420489]]`

`print(a == b)`	`array([[ True, True, True],`
	`[ True, True, True],`
	`[ True, True, True]])`

`def f(x):`
`if x > 5:`
`return x * 2`
`return x * 3`
`vectorized_f = np.vectorize(f)`
`vectorized_f(a)`
La fonction ***vectorize*** permet de rendre f universelle (appliquée à des tableaux entiers plutôt qu'à des éléments individuels).
	Remarques:
	- Les fonctions NumPy sont déjà vectorisées par défaut (appliquées élément par élément à des tableaux NumPy, sans utiliser np.vectorize)
	- Une fonction contenant des structures de contrôle de flux (comme des boucles ou des conditions)doit être vectorisée.
	Résultat:
	`array([[ 3, 6, 9],`
	`[12, 15, 12],`
	`[14, 16, 18]])`

`a.max()` **`print(a.max(axis=0))`** cherche le maximun le long de l'axe 0 **`print(a.max(axis=1))`** cherche le maximun le long de l'axe 1	9 Résultat : obtenir le maximum pour chaque colonne du tableau. [7 8 9] Résultat : obtenir le maximum pour chaque ligne du tableau. [3 6 9]
`a.min();`	1

a.prod();	362880
a.mean() #moyenne arithmétique	5.0
a.sum() a.sum(axis=0) a.sum(axis=1)	45 array([12, 15, 18]) # la somme de chaque colonne array([6, 15, 24]) # la somme de chaque ligne
np.array_equal(a,b)	True

Tri des tableaux

a.sort()	Trier le tableau *a* sur place
np.sort(a)	Renvoie une copie triée du tableau a (l'original n'est donc pas modifié).

Autres manipulations

np.any(a>90)	Teste si le tableau a contient au moins un élément > 90
np.all(a>10)	Teste si tous les éléments > 10
np.nonzero(a)	Renvoie le tableau des positions des éléments non nuls du tableau a.
np.count_nonzero(a)	Compte le nombre d'éléments non nuls du tableau a
np.transpose(a)	Transposé de a
x = np.dot(a, b)	Produit de matrices

np.linalg Module d'algèbre linéaire basique

Soit la matrice: x = np.array([[2, 1],
[7, 4]])

np.linalg.inv(x)	Matrice inverse de x array([[4., -1.], [-7., 2.]])
D, V = np.linalg.eig(x) D : valeurs propres V : vecteurs propres	D = array([0.17157288, 5.82842712]) V = array([[-0.47984149, -0.25272473], [0.8773552 , -0.96753822]])

Supposons que nous cherchons à résoudre le système d'équations linéaires suivant :

```
 2x + y -  z = 8
-3x - y + 2z = -11
-2x + y + 2z = -3

# Coefficients de la matrice A
A = np.array([[ 2,  1, -1],
              [-3, -1,  2],
              [-2,  1,  2]])
# Coefficients de la matrice B
B = np.array([8, -11, -3])

# Résoudre le système linéaire AX = B
solution = np.linalg.solve(A, B)

print(solution) # array([ 2.,  3., -1.])
```

Traçage des courbes : Module pyplot de la bibliothèque Matplotlib

Matplotlib (https://matplotlib.org) est une bibliothèque de visualisation en Pythonest, qui permet de créer des graphiques, des diagrammes et d'autres visualisations de données de manière efficace et flexible.

Pyplot est un module spécifique de Matplotlib qui simplifie la création de graphiques et la visualisations de données.

Dans cette section on donne un bref aperçu des possibilités graphiques de Matplotlib. Il est possible de générer des graphiques à deux et à trois dimensions, et d'agir facilement sur les attributs du graphe (couleurs, type de ligne, axes, annotations...). Les commandes présentées ci-dessous sont utiles notamment pour être introduites dans des scripts et automatiser la production de figure.

```
import matplotlib.pyplot as plt
```

```
plt.plot(X,Y)   permet de tracer une courbe reliant les points dont les
                abscisses sont données dans le vecteur X et les ordonnées
                dans le vecteur Y.
```

Exemple:

```
import matplotlib.pyplot as plt
import numpy as np
from math import pi

x= np.arange(0,2*pi,pi/16)
# ou x= np.linspace(0,2*pi,100) : échantillonner la variable x

plt.plot(x,np.sin(x),label="sin") # on utilise la fonction sinus de Numpy

# Imposer une échelle identique sur les deux axes de coordonnées.
plt.axis("equal")

plt.xlim(-2,10) # Fixer l'intervalle de visualisation sur l'axe des abscisses.

plt.ylabel("fonction sinus")
plt.xlabel("l'axe des abscisses")

plt.title("Fonction sinus")

# sauvegarder la figure (en .png
ou .pdf),
plt.savefig("ma_figure.png")

# pour la voir ou la conserver
plt.show() #Afficher l'image
```

Si tout se passe bien, une fenêtre doit s'ouvrir avec cette figure.