

Manipulation des vecteurs avec numpy

Création, accès, extraction, calculs

Ricco Rakotomalala

http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_python.html

- Numpy est un package pour Python spécialisé dans la manipulation des tableaux (array), pour nous essentiellement les vecteurs et les matrices
- Les tableaux « numpy » ne gère que les objets de même type
- Le package propose un grand nombre de routines pour un accès rapide aux données (ex. recherche, extraction), pour les manipulations diverses (ex. tri), pour les calculs (ex. calcul statistique)
- Les tableaux « numpy » sont plus performants (rapidité, gestion de la volumétrie) que les collections usuelles de Python
- Les tableaux « numpy » sont sous-jacents à de nombreux packages dédiés au calcul scientifique sous Python.
- Attention, un vecteur est en réalité une matrice à 1 seule dimension

Il n'est pas possible de tout aborder dans ce support. Pour aller plus loin, voir absolument le manuel de référence (utilisé pour préparer ce diaporama).

<http://docs.scipy.org/doc/numpy/reference/index.html>

Création à la volée, génération d'une séquence, chargement à partir d'un fichier

CRÉATION D'UN VECTEUR

Préalable important :
importer le module
« numpy »

```
import numpy as np
```

np sera l'alias utilisé
pour accéder aux
routines de la librairie
« numpy ».

Création manuelle à
partir d'un ensemble de
valeurs

```
a = np.array([1.2,2.5,3.2,1.8])
```

Noter le rôle des **[]** pour
délimiter les valeurs

Informations sur la
structure

```
#type de la structure
```

```
print(type(a)) #<class 'numpy.ndarray'>
```

```
#type des données
```

```
print(a.dtype) #float64
```

```
#nombre de dimensions
```

```
print(a.ndim) #1 (on aura 2 si matrice, etc.)
```

```
#nombre de lignes et col
```

```
print(a.shape) #(4,) → on a tuple ! 4 cases sur la 1ère dim (n°0)
```

```
#nombre totale de valeurs
```

```
print(a.size) #4, nb.lignes x nb.colonnes si matrice
```

#création et typage implicite

```
a = np.array([1,2,4])  
print(a.dtype) #int32
```

#création et typage explicite – **préférable !**

```
a = np.array([1,2,4],dtype=float)  
print(a.dtype) #float64  
print(a) #[1. 2. 4.]
```

#un vecteur de booléens est tout à fait possible

```
b = np.array([True,False,True,True], dtype=bool)  
print(b) #[True False True True]
```

la donnée peut être un objet python

```
a = np.array([{"Toto":(45,2000)},{"Tata":(34,1500)})  
print(a.dtype) #object
```

Le typage des valeurs
peut être implicite ou
explicite

Création d'un array d'objets
complexes (autres que les
types de base) est possible

#suite arithmétique de raison 1

```
a = np.arange(start=0,stop=10)
```

```
print(a) #[0 1 2 3 4 5 6 7 8 9], attention dernière valeur est exclue
```

#suite arithmétique de raison step

```
a = np.arange(start=0,stop=10,step=2)
```

```
print(a) #[0 2 4 6 8]
```

#suite linéaire, nb. de valeurs est spécifié par **num**

```
a = np.linspace(start=0,stop=10,num=5)
```

```
print(a) #[0. 2.5 5. 7.5 10.], la dernière valeur est incluse ici
```

#vecteur de valeurs identiques 1 – 1 seule dim et 5 valeurs

```
a = np.ones(shape=5)
```

```
print(a) # [1. 1. 1. 1. 1.]
```

#plus généralement – répétition 5 fois (1 dimension) de la valeur 3.2

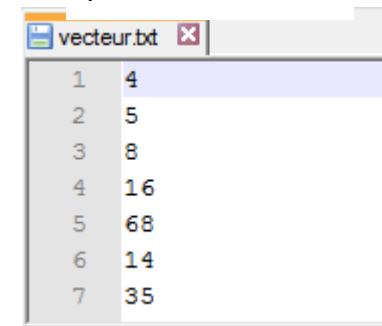
```
a = np.full(shape=5,fill_value=3.2)
```

```
print(a) #[3.2 3.2 3.2 3.2 3.2]
```

Les données peuvent être stockées dans un fichier texte (`loadtxt` pour charger, `savetxt` pour sauver)

```
#charger à partir d'un fichier
#typage explicite possible
a = np.loadtxt("vecteur.txt",dtype=float)
print(a) #[4.  5.  8. 16. 68. 14. 35.]
```

1 seule colonne
pour l'instant



1	4
2	5
3	8
4	16
5	68
6	14
7	35

Remarque : si besoin, modifier le répertoire par défaut avec la fonction `chdir()` du module `os` (qu'il faut importer au préalable)

```
#liste de valeurs
lst = [1.2,3.1,4.5]
print(type(lst)) #<class 'list'>
#conversion à partir d'une liste
a = np.asarray(lst,dtype=float)
print(type(a)) #<class 'numpy.ndarray'>
print(a) #[1.2  3.1  4.5]
```

Conversion d'une collection (type standard Python) en type array de « numpy »

Rajouter une valeur en dernière position

```
#vecteur de valeurs
a = np.array([1.2,2.5,3.2,1.8])
#ajouter une valeur , placée en dernière position
a = np.append(a,10)
print(a) #[1.2 2.5 3.2 1.8 10.]
```

Suppression d'une valeur via son indice

```
#suppression via indice
b = np.delete(a,2) #une plage d'indices est aussi possible
print(b) #[1.2 2.5 1.8 10.]
```

Redimensionnement explicite

```
a = np.array([1,2,3])
#redimensionnement
#1 valeur pour vecteur, couple de valeurs pour matrice
a.resize(new_shape=5)
print(a) #[1 2 3 0 0], les nouvelles cases mises à 0
```

Concaténation de 2 vecteurs

```
#concatenation 2 vecteurs
x = np.array([1,2,5,6])
y = np.array([2,1,7,4])
z = np.append(x,y)
print(z) #[1 2 5 6 2 1 7 4]
```


Accéder aux valeurs via des indices ou des conditions

EXTRACTION DES VALEURS

#toutes les valeurs

```
print(v)
```

#ou

```
print(v[:]) # noter le rôle du :, il faut lire ici début à fin
```

#accès indicé - première valeur

```
print(v[0]) # 1.2 – Noter que le 1er indice est 0 (zéro)
```

#dernière valeur

```
print(v[v.size-1]) #6.3, v.size est ok parce que v est un vecteur
```

#plage d'indices contigus

```
print(v[1:3]) # [7.4 4.2]
```

#extrêmes, début à 3 (non-inclus)

```
print(v[:3]) # [1.2 7.4 4.2]
```

#extrêmes, 2 à fin

```
print(v[2:]) # [4.2 8.5 6.3]
```

#indice négatif

```
print(v[-1]) # 6.3, dernier élément
```

#indices négatifs

```
print(v[-3:]) # [4.2 8.5 6.3], 3 derniers éléments
```

Remarque : Mis à part les singletons, les vecteurs générés sont de type `numpy.ndarray`

La notation générique des indices est : `début:fin:pas`
`fin` est non inclus dans la liste

`#valeur n°1 à n°3 avec un pas de 1`
`print(v[1:4:1]) # [7.4, 4.2, 8.5]`

`#le pas de 1 est implicite`
`print(v[1:4]) # [7.4, 4.2, 8.5]`

`#n°0 à n°2 avec un pas de 2`
`print(v[0:3:2]) # [1.2, 4.2]`

`#le pas peut être négatif, n°3 à n°1 avec un pas de -1`
`print (v[3:0:-1]) # [8.5, 4.2, 7.4]`

`#on peut exploiter cette idée pour inverser un vecteur`
`print(v[::-1]) # [6.3, 8.5, 4.2, 7.4, 1.2]`

```
#extraction avec un vecteur de booléens  
#si b trop court, tout le reste est considéré False  
b = np.array([False,True,False,True,False],dtype=bool)  
print(v[b]) # [7.4 8.5]
```

```
#on peut utiliser une condition pour l'extraction  
print(v[v < 7]) # [1.2 4.2 6.3]
```

```
#parce que la condition est un vecteur de booléen  
b = v < 7  
print(b) # [True False True False True]  
print(type(b)) # <class 'numpy.ndarray'>
```

```
#on peut utiliser la fonction extract()  
print(np.extract(v < 7, v)) # [1.2 4.2 6.3]
```

```
#recherche valeur max
```

```
print(np.max(v)) # 8.5
```

```
#recherche indice de valeur max
```

```
print(np.argmax(v)) # 3
```

```
#tri des valeurs
```

```
print(np.sort(v)) # [1.2 4.2 6.3 7.4 8.5]
```

```
#récupération des indices triés
```

```
print(np.argsort(v)) # [0 2 4 1 3]
```

```
#valeurs distinctes
```

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

```
print(np.unique(a)) # [1 2]
```

Remarque : L'équivalent existe pour min()

CALCULS SUR LES VECTEURS

`#moyenne`

`print(np.mean(v)) # 5.52`

`#médiane`

`print(np.median(v)) # 6.3`

`#variance`

`print(np.var(v)) # 6.6856`

`#quantile`

`print(np.percentile(v,50)) #6.3 (50% = médiane)`

`#somme`

`print(np.sum(v)) # 27.6`

`#somme cumulée`

`print(np.cumsum(v)) # [1.2 8.6 12.8 21.3 27.6]`



La librairie n'est pas très fournie, nous aurons besoin de SciPy (et autres)

```
#calculs entre vecteurs
x = np.array([1.2,1.3,1.0])
y = np.array([2.1,0.8,1.3])
#multiplication
print(x*y) # [2.52 1.04 1.3]
#addition
print(x+y) # [3.3 2.1 2.3]
#multiplication par un scalaire
print(2*x) # [2.4 2.6 2.]
```

#comparaison de vecteurs

```
x = np.array([1,2,5,6])
y = np.array([2,1,7,4])
b = x > y
print(b) # [False True False True]
```

#opérations logiques

```
a = np.array([True,True,False,True],dtype=bool)
b = np.array([True,False,True,False],dtype=bool)
#ET logique
np.logical_and(a,b) # [True False False False]
#OU exclusif logique
np.logical_xor(a,b) # [False True True True]
```

Principe : Les calculs se font élément par élément (elementwise) entre vecteurs « numpy » - On a le même principe sous R.

La liste des fonctions est longue.
Voir -
<http://docs.scipy.org/doc/numpy/reference/routines.logic.html>


```
x = np.array([1.2,1.3,1.0])  
y = np.array([2.1,0.8,1.3])
```

```
#produit scalaire
```

```
z = np.vdot(x,y)  
print(z) # 4.86
```

```
#ou l'équivalent calculé
```

```
print(np.sum(x*y)) # 4.86
```

```
#norme d'un vecteur
```

```
n = np.linalg.norm(x)  
print(n) # 2.03
```

```
#ou l'équivalent calculé
```

```
import math  
print(math.sqrt(np.sum(x**2))) # 2.03
```

Principe : Des fonctions
spécifiquement matricielles
existent, certaines s'appliquent à
des vecteurs

#opérations ensemblistes

```
x = np.array([1,2,5,6])  
y = np.array([2,1,7,4])
```

#intersection

```
print(np.intersect1d(x,y)) # [1 2]
```

#union – attention, ce n'est pas une concaténation

```
print(np.union1d(x,y)) # [1 2 4 5 6 7]
```

#différence c.à-d. qui sont dans x et pas dans y

```
print(np.setdiff1d(x,y)) # [5 6]
```

Principe : Un vecteur de valeurs (surtout entières) peut être considéré comme un ensemble de valeurs.

De la documentation à profusion (n'achetez pas des livres sur Python)

Site du cours

http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_python.html

Site de Python

Welcome to Python - <https://www.python.org/>

Python 3.4.3 documentation - <https://docs.python.org/3/index.html>

Portail Python

Page Python de [Developpez.com](http://developpez.com)

Quelques cours en ligne

P. Fuchs, P. Poulain, « [Cours de Python](#) » sur Developpez.com

G. Swinnen, « [Apprendre à programmer avec Python](#) » sur Developpez.com

« [Python](#) », Cours interactif sur [Codecademy](#)

POLLS (KDnuggets)

Data Mining / Analytics Tools Used

Python, 4^{ème} en [2015](#)

What languages you used for data mining / data science?

Python, 3^{ème} en [2014](#) (derrière R et SAS)