Programmation en Python (VII)

Akka Zemmari
Hervé Hocquard
herve.hocquard@u-bordeaux.fr

LaBRI, Université de Bordeaux - CNRS

17 septembre, 2023

université de BORDEAUX

Vecteurs avec Numpy

Vecteurs avec Numpy

En prévision d'un prochain cours sur l'IA ...

Numpy

- 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 gèrent 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)

université de BORDEAUX

Numpy

- 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.
- Un vecteur est en réalité une matrice à 1 seule dimension
- voir `http://docs.scipy.org/doc/numpy/reference/index.html`

université de BORDEAUX

Numpy

Création d'un vecteur :

```
#On importe d'abord le package
import numpy as np

#On peut créer le vecteur "manuellement"
v = np.array([1.2, 2.5, 3.2, 1.8])
```

Numpy

Informations sur la structure :

```
1  #type de la structure
2  print(type(v)) #<class 'numpy.ndarray'>
3
4  #type des données
5  print(v.dtype) #float64
6
7  #nombre de dimensions
8  print(v.ndim)  #1 (on aura 2 si matrice, etc.)
9
10 #nombre de lignes et de colonnes
11 print(v.shape) #(4,)--> on a un tuple ! 4 cases sur la 1ère dim (n°0)
12
13 #nombre totale de valeurs
14 print(v.size) #4, nb.lignes * nb.colonnes si matrice
```

Numpy

Typage des données : implicite ou explicite

```
1  #création et typage implicite
2  a = np.array([1,2,4])
3  print(a.dtype) #int32
4
5  #création et typage explicite, préférable !!
6  a = np.array([1,2,4],dtype=float)
7  print(a.dtype) #float64
8
9  print(a) #[1. 2. 4.]
10
11 #un vecteur de booléens est tout à fait possible
12 b = np.array([True,False,True,True], dtype=bool)
13 print(b) #[True False True True]
```

Numpy

Création d'une séquence de valeurs

```
1  #suite arithmétique de raison 1
2  a = np.arange(start=0,stop=10)
3  print(a) #[0 1 2 3 4 5 6 7 8 9], attention la dernière valeur est exclue
4
5  #suite arithmétique de raison step
6  a = np.arange(start=0,stop=10,step=2)
7  print(a) #[0 2 4 6 8]
8
9  #suite linéaire, nb. de valeurs est spécifié par num
10 a = np.linspace(start=0,stop=10,num=5)
11 print(a) #[0. 2.5 5. 7.5 10.], la dernière valeur est incluse ici
12
13 #vecteur de valeurs identiques 1, 1 seule dim et 5 valeurs
14 a = np.ones(shape=5)
15 print(a) # [1. 1. 1. 1. 1.]
```

Numpy

Création d'une séquence de valeurs

```
1  #plus généralement, répétition 5 fois (1 dimension) de la valeur 3.2
2  a = np.full(shape=5,fill_value=3.2)
3  print(a) #[3.2 3.2 3.2 3.2 3.2]
4
5  #plus généralement, répétition (nb lignes * nb colonnes) fois de la valeur 3.2
6  a = np.full((4,6),fill_value=3.2)
7  print(a) # [[3.2 3.2 3.2 3.2 3.2]
8           #  [3.2 3.2 3.2 3.2 3.2]
9           #  [3.2 3.2 3.2 3.2 3.2]
10          #  [3.2 3.2 3.2 3.2 3.2]]
```

université de BORDEAUX

Numpy

Création d'une séquence de valeurs

```
1  #matrice de valeurs identiques 0, sur nb lignes et nb colonnes
2  a = np.zeros((4,5))
3  print(a) # [[0. 0. 0. 0. 0.]
4           #  [0. 0. 0. 0. 0.]
5           #  [0. 0. 0. 0. 0.]
6           #  [0. 0. 0. 0. 0.]]
7
8  #matrice de valeurs identiques 1, sur nb lignes et nb colonnes
9  a = np.ones((4,5),dtype=int)
10 print(a) # [[1 1 1 1 1]
11          #  [1 1 1 1 1]
12          #  [1 1 1 1 1]
13          #  [1 1 1 1 1]]
```

Numpy

Création d'une séquence de valeurs "aléatoires"

```
1  #matrice de valeurs aléatoires de type entier sur nb lignes et nb colonnes
2  a = np.empty((4,5),dtype=int)
3  print(a) # [[1886220131     583998  293208064 1868787273 1667592818 1634738292]
4           #  [1852400740    4289127   41091072  805306368     127793 1917779968]
5           #  [-70           -37000    3895808  377094144 762212174 -1702060386]
6           #  [1679831603 1953064809 1970234912 1030906990 1040187392          5]]
7
8  #vecteur de valeurs aléatoires de type float sur nb colonnes
9  a = np.empty(shape=3)
10 print(a) # [-4.47997762e-305 4.49124206e-305 6.20184400e-317]
```

Numpy

Chargement à partir d'un fichier - Conversions

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

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

université de BORDEAUX

Numpy

Redimensionnement

```
1  #vecteur de valeurs
2  a = np.array([1.2,2.5,3.2,1.8])
3
4  #ajouter une valeur , placée en dernière position
5  a = np.append(a,10)
6  print(a) #[1.2 2.5 3.2 1.8 10.]
7
8  #suppression via indice
9  b = np.delete(a,2) #une plage d'indices est aussi possible
10 print(b) #[1.2 2.5 1.8 10.]
```

université de BORDEAUX

Numpy

Redimensionnement

```
1  a = np.array([1,2,3])
2  #redimensionnement
3  #1 valeur pour vecteur, couple de valeurs pour matrice
4  a.resize(new_shape,refcheck=False) #on remplace new_shape par 5 ici
5  #ou alors on écrit sur la ligne de dessus new_shape=5
6  #on aurait pu aussi écrire a.resize((1,5),refcheck=False)
7  print(a) #[1 2 3 0 0], les nouvelles cases mises à 0...
8
9  #concatenation de 2 vecteurs
10 x = np.array([1,2,5,6])
11 y = np.array([2,1,7,4])
12 z = np.append(x,y)
13 print(z) #[1 2 5 6 2 1 7 4]
```

Numpy

Extraction des valeurs

```
1  v = np.array([1.2,7.4,4.2,8.5,6.3])
2  #toutes les valeurs
3  print(v)
4  #ou
5  print(v[:]) # noter le rôle du : , il faut lire ici début à fin
6
7  #accès indicé
8  #première valeur
9  print(v[0]) # 1.2 - Noter que le 1er indice est 0 (zéro)
10
11 #dernière valeur
12 print(v[v.size-1]) #6.3, v.size est ok parce que v est un vecteur
```

Numpy

Accès indicé - Plages d'indices

```
1  #plage d'indices contigus
2  print(v[1:3]) # [7.4 4.2]
3
4  #extrêmes, début à 3 (non-inclus)
5  print(v[:3]) # [1.2 7.4 4.2]
6
7  #extrêmes, 2 à fin
8  print(v[2:]) # [4.2 8.5 6.3]
9
10 #indice négatif
11 print(v[-1]) # 6.3, dernier élément
12
13 #indices négatifs
14 print(v[-3:]) # [4.2 8.5 6.3], 3 derniers éléments
```

Numpy

Accès indicé – Ecriture générique

La notation générique des indices est : *debut:fin:pas*,

fin est non inclus dans la liste

Numpy

Accès indicé – Écriture générique

```
1  v = np.array([1.2,7.4,4.2,8.5,6.3])
2
3  #valeur n°1 à n°3 avec un pas de 1
4  print(v[1:4:1]) # [7.4, 4.2, 8.5]
5
6  #le pas de 1 est implicite
7  print(v[1:4]) # [7.4, 4.2, 8.5]
8
9  #n°0 à n°2 avec un pas de 2
10 print(v[0:3:2]) # [1.2, 4.2]
11
12 #le pas peut être négatif, n°3 à n°1 avec un pas de -1
13 print (v[3:0:-1]) # [8.5, 4.2, 7.4]
14
15 #on peut exploiter cette idée pour inverser un vecteur
16 print(v[::-1]) # [6.3, 8.5, 4.2, 7.4, 1.2]
```

Numpy

Accès par conditions – Indiçage booléen

```
1  v = np.array([1.2,7.4,4.2,8.5,6.3])
2
3  #extraction avec un vecteur de booléens
4  #si b trop court, tout le reste est considéré False
5  b = np.array([False,True,False,True,False],dtype=bool)
6  print(v[b]) # [7.4 8.5]
7
8  #on peut utiliser une condition pour l'extraction
9  print(v[v < 7]) # [1.2 4.2 6.3]
10
11 #parce que la condition est un vecteur de booléens
12 b = v< 7
13 print(b) # [True False True False True]
14 print(type(b)) # <class 'numpy.ndarray'>
15
16 #on peut utiliser la fonction extract()
17 print(np.extract(v < 7, v)) # [1.2 4.2 6.3]
```

Numpy

Tri et recherche

```
1  v = np.array([1.2,7.4,4.2,8.5,6.3])
2
3  #recherche valeur max
4  print(np.max(v)) # 8.5
5
6  #recherche indice de valeur max
7  print(np.argmax(v)) # 3
8
9  #tri des valeurs
10 print(np.sort(v)) # [1.2 4.2 6.3 7.4 8.5]
11
12 #récupération des indice triés
13 print(np.argsort(v)) # [0 2 4 1 3]
14
15 #valeurs distinctes
16 a = np.array([1,2,2,1,1,2])
17 print(np.unique(a)) # [1 2]
```

Numpy

Calculs (statistiques) récapitulatifs

```
1  v = np.array([1.2,7.4,4.2,8.5,6.3])
2
3  #moyenne
4  print(np.mean(v)) # 5.52
5
6  #médiane
7  print(np.median(v)) # 6.3
8
9  #variance
10 print(np.var(v)) # 6.6856
11
12 #quantile
13 print(np.percentile(v,50)) #6.3 (50% = médiane)
14
15 #somme
16 print(np.sum(v)) # 27.6
17
18 #somme cumulée
19 print(np.cumsum(v)) # [1.2 8.6 12.8 21.3 27.6]
```

Numpy

Calculs entre vecteurs – Le principe « elementwise »

```
1  #calculs entre vecteurs
2  x = np.array([1.2,1.3,1.0])
3  y = np.array([2.1,0.8,1.3])
4
5  #multiplication
6  print(x*y) # [2.52 1.04 1.3]
7
8  #addition
9  print(x+y) # [3.3 2.1 2.3]
10
11 #multiplication par un scalaire
12 print(2*x) # [2.4 2.6 2. ]
```

Numpy

Calculs entre vecteurs – Le principe « elementwise »

```
1  #comparaison de vecteurs
2  x = np.array([1,2,5,6])
3  y = np.array([2,1,7,4])
4  b = x > y
5  print(b) # [False True False True]
6
7  #opérations logiques
8  a = np.array([True,True,False,True],dtype=bool)
9  b = np.array([True,False,True,False],dtype=bool)
10
11 #ET logique
12 np.logical_and(a,b) # [True False False False]
13
14 #OU exclusif logique
15 np.logical_xor(a,b) # [False True True True]
```

Numpy

Fonctions matricielles

```
1  x = np.array([1.2,1.3,1.0])
2  y = np.array([2.1,0.8,1.3])
3
4  #produit scalaire
5  z = np.vdot(x,y)
6  print(z) # 4.86
7
8  #ou l'équivalent calculé
9  print(np.sum(x*y)) # 4.86
10
11 #norme d'un vecteur
12 n = np.linalg.norm(x)
13 print(n) # 2.03
14
15 #ou l'équivalent calculé
16 import math
17 print(math.sqrt(np.sum(x**2))) # 2.03
```

Numpy

Opérations ensemblistes

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

```
1  x = np.array([1,2,5,6])
2  y = np.array([2,1,7,4])
3
4  #intersection
5  print(np.intersect1d(x,y)) # [1 2]
6
7  #union - attention, ce n'est pas une concaténation
8  print(np.union1d(x,y)) # [1 2 4 5 6 7]
9
10 #différence c.à-d. qui sont dans x et pas dans y
11 print(np.setdiff1d(x,y)) # [5 6]
```