Algorithmique

Présentation de la bibliothèque Numpy

Anicet E. T. Ebou, ediman.ebou@inphb.ci

Bibliothèque NumPy

- NumPy est une bibliothèque Python utilisée pour travailler avec des tableaux.
- Elle dispose également de fonctions permettant de travailler dans le domaine de l'algèbre linéaire, de la transformée de Fourier et des matrices.
- NumPy a été créé en 2005 par Travis Oliphant. Il s'agit d'un projet open source utilisable librement. NumPy est l'acronyme de Numerical Python.

Bibliothèque NumPy

- En Python, nous avons des listes qui servent de tableaux, mais elles sont lentes à traiter.
- NumPy vise à fournir un objet tableau qui est jusqu'à 50 fois plus rapide que les listes traditionnelles de Python.

Création d’un tableau

```
# Chargement du module

import numpy as np

liste = [1 ,2 ,3 ,4 ,5]

tab = np.array(liste) # crée un tableau à partir de la liste

type(tab)             # numpy.ndarray

tab.dtype             # int64: type des éléments du tableau

# matrice de taille 2 x 3

matrice = np.array([[1 ,0 ,0], [0 ,1 ,0]])

# erreur: des types différents

tab = np.array([1 , ’a’, 2])
```

Commandes arange et linspace

La commande `arange` fonctionne comme la commande `range` mais pour créer un tableau. La commande `linspace` renvoie des nombres uniformément espacés sur un intervalle spécifié.

```
tab = np.array(range(100))
tab = np.arange(100)     # même résultat
tab = np.arange(-5, 6)
tab = np.arange(-5, 6.4,.2) # arange accepte des arguments non entier
tab = np.linspace(-5, 6, 100) #le 3e argument est le nombre de points
```

Passage des tableaux aux listes

```
matrice.tolist() # crée une liste à partir du tableau

type(matrice)  # renvoie numpy.ndarray car matrice n’est pas modifiée

matrice = matrice.tolist()    # remplace le tableau par une liste .

matrice = array(matrice)      # revient au tableau
```

Attributs d’un tableau

```
matrice.dtype  # type des éléments du tableau

matrice.shape  # dimension sous la forme d’un tuple

matrice.shape[0] # nombre de lignes d’une matrice

matrice.shape[1] # nombre de colonnes d’une matrice

matrice.size   # nombre total de coefficients

matrice.ndim   # nombre de dimensions du tableau (2 pour matrice)
```

Lecture-écriture

```
tab = np.arange(10)

matrice = np.array([[10 * j + i  for i in range(10)] for j in range(8)])

tab[4]              # élément d'indice 4 dans le tableau

matrice[4]          # tableau de la ligne d'indice 4

matrice[4, :]       # idem

matrice[4, 3]       # élément d'indices 4,3

matrice[5:, 5:]     # bloc inférieur droit

matrice[:: -1]      # inverse l'ordre des lignes
```

Lecture-écriture

```
lst = [[10 * j + i for i in range(10)] for j in range(8)]
lst[5][2]      # dans le tableau lst[5], je demande l'élément 2
matrice = np.array(lst)
matrice[5, 2]  # je demande le coefficient de coordonnées 5,2
matrice[5][2]  # Numpy est gentil: fonctionne aussi
tab[4] = 20    # modifie l'élément d'indice 4
tab[4] = 'a'   # erreur, pas le bon type
tab[4] = 19.5  # remplace par int(19.5) = 19
```

Lecture-écriture

```
matrice[5 ,7] = 12

matrice[:5 ,:5] = 0       # remplace le bloc 5 x 5 en haut à gauche par des 0

matrice[:2 ,1] = [-1, -2]  # remplace les 2 premières coordonnées de la col 1

matrice[:2 ,1] = [1 ,2 ,3] # erreur, mauvaise dimension
```

Gestion mémoire des tableaux

Il faut faire attention aux effets de gestion de la mémoire dans Python. Contrairement aux listes, lorsque vous utilisez le slicing, Python ne recopie pas la partie du tableau concernée, mais fait pointer une nouvelle variable vers un extrait du même tableau.

Si vous modifiez le contenu de cette nouvelle variable, cela modifiera aussi le tableau initial.

La méthode `copy()` permet de faire une copie complète du tableau.

Gestion mémoire des tableaux

```
liste = [0 ,1 ,2 ,3 ,4]

liste2 = liste

liste2 is liste          # True: même objet mémoire

liste3 = liste[:]

liste3 is liste          # False: nouvel objet mémoire

liste4 = liste.copy()

liste4 is liste          # False: nouvel objet mémoire
```

Gestion mémoire des tableaux

```
tab2 = tab

tab2 is tab          # True : même objet mémoire

tab3 = tab[:]

tab3 is tab          # False : même objet mémoire

tab4 = tab.copy()

tab4 is tab
```

Création des tableaux usuels

Ils sont construits par la fonction `zeros(taille)` où taille est un tuple qui contient les dimensions.

Par défaut, les éléments du tableau sont de type `float` (modifiable avec un argument optionnel).

```
np.zeros(10)   # tableau à une dimension de taille 10 rempli de 0
np.zeros((3, 5))  # taille 3 x 5 ne pas oublier les doubles parenthèses
np.zeros((3, 5), int)  # de type entier
```

Création des tableaux usuels

```
np.ones(10)              # tableau à une dimension de taille 10 rempli de 1
np.ones((3, 5))          # taille 3 x5 rempli de 1
5 * np.ones((3, 5))      # taille 3 x5 rempli de 5
np.ones ((3, 5), int )   # de type entier
# matrice identité
np.eye(4)                # identité de taille 4 x 4
np.eye(4, dtype = int)   # identité de taille 4 x 4, de type int
np.eye(4, k = 1)         # surdiagonale avec des 1
np.eye(4, k = -1)        # sousdiagonale avec des 1
```

Création des tableaux usuels

```
# matrices diagonales
np.diag([1 ,1 ,1])  # identité de taille 3 x3
np.diag([1 ,2 ,3 ,4]) # taille 4 x 4 avec 1, 2, 3, 4 en diagonale
np.diag([1 ,2 ,3], k =1) # taille 4 x 4, surdiagonale avec 1, 2, 3
np.diag([1 ,2 ,3], k = -1) # sousdiagonale
# matrice triangulaire inférieure
np.tri(3)
```