|  |  |  |
| --- | --- | --- |
| **UNIVERSITE BADJI Mokhtar– ANNABA**  **FACULTE DES SCIENCES DE L’INGENIORAT**  **DEPARTEMENT D’INFORMATIQUE** |  | Intelligence Artificielle et Traitement de l’information **(IATI)**  Méthodes de Prétraitement de l’Information **(MPI)**  Réalisé par **: A. Melouah** |

Manipulation des tableaux et matrices avec Numpy

1. **Introduction**

* Numpy est un package pour Python spécialisé dans la manipulation des tableaux (array) 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)
* Une matrice est un tableau (array) à 2 dimensions
* Pour utiliser les tableaux de Numpy il faut l’importer par

import numpy as np

1. **Création de matrices**

|  |  |  |  |
| --- | --- | --- | --- |
| **Commande** | **Permet de** | **Exemple** | **Résultat** |
| array | Permet de définir une matrice | A = np.array([[1, 2, 3], [4, 5, 6]]) | array [[1, 2, 3],  [4, 5, 6]]) |
| zeros | permet de créer des matrices remplies de 0 | A=np.zeros((2,3)) | array([[ 0., 0., 0.],  [ 0., 0., 0.]]) |
| ones | permet de créer des matrices remplies de 1 | A=np.ones((3,2)) | array ([[ 1., 1.],  [ 1., 1.],  [ 1., 1.]]) |
| diag | permet de créer une matrice diagonale | A=np.diag([1,2,3]) | array([[1, 0, 0],  [0, 2, 0],  [0, 0, 3]]) |
| eyes | Permet de créer une matrice du type 𝐼*n* où 𝑛 est un entier | A=np.eye(4) | array([[ 1., 0., 0., 0.],  [ 0., 1., 0., 0.],  [ 0., 0., 1., 0.],  [ 0., 0., 0., 1.]]) |
| full | permet de créer une matrice et de la remplir | A = np.full(shape=(2,4), fill\_value=0.1) |  |

L’attribut shape donne la taille d’une matrice : nombre de lignes, nombre de colonnes. On peut redimensionner une matrice, sans modifier ses termes, à l’aide de la méthode reshape.

A.shape

(2, 3)

A = A.reshape((3, 2))

A

array([[1, 2],

[3, 4],

[5, 6]])

1. **Accès aux termes de la matrice**

|  |  |  |
| --- | --- | --- |
| **Opération** | **Sur** | **Exemple** |
| L’accès à un terme de la matrice | A[i, j]  i désigne la ligne et  j la colonne. | A[1, 0] # terme de la deuxième ligne, première colonne  **Attention, les indices commencent à zéro !** |
| Slicing | ligne | A[0, :] # première ligne sous forme de tableau à 1 dimension  array([1, 2])  A[0, :].shape  (2,)  A[0 :1, :] # première ligne sous forme de matrice ligne  array([[1, 2]])  A[0 :1, :].shape  (1, 2) |
| colonne | A[ :, 1] # deuxième colonne sous forme de tableau à 1 dimension  array([2, 4, 6])  A[ :, 1 :2] # deuxième colonne sous forme de matrice colonne  array([[2],  [4],  [6]]) |
| sous-matrice | A[1 :3, 0 :2] # sous-matrice lignes 2 et 3, colonnes 1 et 2  array([[3, 4],  [5, 6]]) |

1. **Fonctions sur les matrices**

|  |  |  |
| --- | --- | --- |
| **Fonction** | **Permet de** | **Exemple** |
| concatenate | permet de créer des matrices par blocs | A = np.ones((2,3))  B = np.zeros((2,3))  np.concatenate((A,B), axis=0)  array([[ 1., 1., 1.],  [ 1., 1., 1.],  [ 0., 0., 0.],  [ 0., 0., 0.]]) |
|  | np.concatenate((A,B),axis=1)  array([[ 1., 1., 1., 0., 0., 0.],  [ 1., 1., 1., 0., 0., 0.]]) |
| + et \* | Les opérations d’ajout et de multiplication par un scalaire se font avec les opérateurs + et \*. | A = np.array([[1,2], [3,4]])  B = np.eye(2)  A + 3\*B  array([[ 4., 2.],  [ 3., 7.]]) |
| dot | Pour effectuer un produit matriciel (lorsque que cela est possible) | A = np.array([[1,2], [3,4]])  B = np.array([[1,1,1], [2,2,2]])  np.dot(A, B)  array([[ 5, 5, 5],  [11, 11, 11]]) |
| On peut également utiliser la méthode dot qui est plus pratique pour calculer un produit de plusieurs matrices. | A.dot(B)  array([[ 5, 5, 5],  [11, 11, 11]])  A.dot(B).dot(np.ones((3,2)))  array([[ 15., 15.],  [ 33., 33.]]) |
| transpose | La transposée s’obtient avec la fonction transpose | np.transpose(B)  array([[1, 2],  [1, 2],  [1, 2]]) |
| L’expression A.T renvoie aussi la transposée de A. | B.T  array([[1, 2],  [1, 2],  [1, 2]]) |