Programmation et Applications Interactives

Introduction à NumPy

Philippe Blasi

Pourquoi NumPy ?

- **Python : langage interprété**
- **Pas conçu au départ pour le calcul numérique**
- **Mais utilisé par la communauté scientifique et technique**
  - Calcul scientifique
  - Data Science
- **Nécessité d’une bibliothèque efficace**

NumPy pour Numerical Python (numpy.org)

- **Bibliothèque efficace écrite en Python, C et Fortran**
- **Libre et Open Source**
- **Multiplate-forme**
- **Destinée à manipuler des tableaux**
  - Multidimensionnels (matrices nd, ndarray)
  - Données homogènes (toutes du même type)
  - Fonctions nombreuses (+500) ⇒ documentation
- **Fonctionnalités comparables à celles de MATLAB**

NumPy pour Numerical Python (numpy.org)

- **Bibliothèques complémentaires**
  - SciPy : fonctionnalités comparables à celles de MATLAB
    - Apprentissage automatique SciKit-Learn et SciKit-Image
  - Matplotlib : représentation graphique
  - OpenCV (Computer Vision) : vision par ordinateur
- **Multiplate-forme**
- **Destinée à manipuler des tableaux**
  - Multidimensionnels (matrices nd, ndarray)
  - Données homogènes (toutes du même type)
  - Fonctions nombreuses (+500) ⇒ documentation

Démarrer avec NumPy

- **Installer la bibliothèque**
  - `pip install numpy`
- **Plus simple, installer Anaconda**
- **Importer la bibliothèque**
  - `import numpy as np`
- **Utiliser la bibliothèque**
  - `arr = np.array([1,2,3,4,5])`

Création de matrices

- **0-D matrice = scalaire**
  - `arr = np.array(66)`
- **1-D matrice = vecteur**
  - `arr = np.array([1,2,3,4,5]) # à partir d'une liste`
- **2-D matrice = matrice**
  - `arr = np.array([[1,2,3],[4,5,6]]) # à partir d'une liste de liste`

Création de matrices

- **Choisir le type des éléments**
  - Paramètre dtype ⇒ documentation
  - Type python
    - `arr = np.array([1,2,3,4,5],dtype=float)`
  - Type numpy
    - `arr = np.array([1,2,3,4,5],dtype=np.float32)`
  - Notation avec une chaîne
    - `arr = np.array([1,2,3,4,5],dtype='float64')`

Création de matrices

- **Propriétés pour une matrice a**
  - a.dtype = type de donnée des éléments
  - a.ndim = nombre de dimensions
  - a.shape = nombre d'éléments pour chaque dimensions (tuple)
  - a.size = nombre total d'éléments de la matrice
  - a.itemsize = nombre d'octets par élément
  - a.nbytes = nombre d'octets pour la matrice

Accès par index

- **1-D matrice : comme en Python**
  - `arr = np.array([1,2,3,4,5])`
  - `print(arr[1])`
- **2-D matrice : une petite différence**
  - `arr = np.array([[1,2,3],[4,5,6]])`
  - `print(arr[1,2])`
- **3-D matrice : etc**
  - `arr = np.array([[[1,2], [3,4]], [[5,6], [7,8]]])`
  - `print(arr[0,1,0])`

Accès par index

- **1-D matrice : presque comme en Python**
  - `arr = np.array([1,2,3,4,5])`
  - `nb_row = arr.shape[0]  # au lieu de len(arr)`
  -
```
for idx_row in range(nb_row):
    print(arr[idx_row])
```

Accès par index

- **2-D matrice**
  - `arr = np.array([[1,2,3],[4,5,6]])`
  - `nb_row, nb_col = arr.shape`
  -
```
for idx_row in range(nb_row):
    for idx_col in range(nb_col):
        print(arr[idx_row,idx_col])
```

Accès par itération sur les éléments (boucle for)

- **1-D matrice : comme en Python**
  - `arr = np.array([1,2,3,4,5])`
  -
```
for val in arr:
  print(val)
```

Accès par itération sur les éléments (boucle for)

- **2-D matrice**
  - `arr = np.array([[1,2,3],[4,5,6]])`
  -
```
for row in arr:
  for val in row:
    print(val)
```

Accès par itération sur les éléments (boucle for)

- **Si aussi besoin des indices**
- **1-D matrice : comme en Python, enumerate**
  - `arr = np.array([1,2,3,4,5])`
  - ```
    for idx,val in enumerate(arr):
      print(f"index={idx},valeur={val}")
    ```
    ```

Accès par itération sur les éléments (boucle for)

- **Si aussi besoin des indices**
- **2-D matrice : presque comme en Python, np.ndenumerate**
  - `arr = np.array([[1,2,3],[4,5,6]])`
  - ```
    for idx,val in np.ndenumerate(arr):
      print(f"index={idx},valeur={val}")
    ```
    ```

Accès par prédicat (condition)

- **Nouvelle fonctionnalité bien plus puissante**
- **Accès à tous les éléments remplissant la condition**
  - Soit en lecture : juste récupération de la valeur
  - Soit en écriture : modification de la valeur

Accès par prédicat (condition)

- **Exemple en lecture**
  - `m = np.array([[1,-2,3],[-4,5,-6]])`
  - `print(m[m>0]) # => [1 3 5]`
  - `print(m[m%2 == 0]) # => [-2 -4 -6]`
  - `print(m[(m>-3) & (m<3)]) # => [1 -2]`
  - `print(m[(m<-3) | (m>3)]) # => [-4  5 -6]`

Accès par prédicat (condition)

- **Exemple en écriture**
  - `m = np.array([[1,-2,3],[-4,5,-6]])`
  - `m[m<0] = 66 # => [[1,66,3],[66,5,66]]`
  - `m[m==66] = 0 # => [[1,0,3],[0,5,0]]`

Accès par prédicat (condition)

- **np.where : équivalent if : else :**
- **Ne modifie pas la matrice**
  - `m = np.array([[1,-2,3],[-4,5,-6]])`
  - `print(np.where(m<0,11, m*2))`
    `# => [[2,11,6],[11,25,11]]`

Découpage des matrices (slicing)

- **Comme en Python**
  - `arr[debut:fin:pas] # pour chaque dimension`
- **Exemples 1D**
  - `arr[1:10]      # de 1 à 10 non compris`
  - `arr[10:1:-1]   # de 10 à 1 non compris par pas de -1`
  - `arr[:10]       # du début à 10 non compris`
  - `arr[2:]        # du 2 à la fin`
  - `arr[::-1]      # de la fin au début`

Découpage des matrices (slicing)

- **Exemples 2D**
  - `arr[1,1:4]` `# ligne 1, colonnes 1 à 4 non compris`
  - `arr[0:2,1:4]` `# lignes 0 et 1, colonnes 1, 2 et 3`
  - `arr[0:2,4:1:-1]` `# lignes 0 et 1, colonnes 4, 3 et 2`
  - `arr[::-1,::-1]` `# tout arr à l'envers`

Enumération

- **Indique dans un tuple les lignes et colonnes à parcourir**
  - `arr = np.array([[1,2,3],[4,5,6]])`
  - `arr[0,(1,0,2)] # ⇒ [2,1,3]`
  - `arr[::-1,(1,0,2)] # ⇒ [[5,4,6][2,1,3]]`

Transformation de matrices

- **Transposée**
  - `arr.T`
- **Conversion en vecteur**
  - `arr.ravel()`
  - `arr.repeat(3)  # idem en répétant chaque élément 3 fois`
- **Changement de taille**
  - `arr.reshape(nb_row,nb_col)`
  - `arr.reshape(dim0,dim1,dim2)`
- **Ajout d'une dimension**
  - `arr[None,:,:]  # en tête`
  - `arr[:,None,:]  # au milieu`
  - `arr[:,:,None]  # à la fin`
-

Transformation de matrices

- **Répétition**
  - `np.tile(arr,[2,3]) # répéter arr 2 fois en hauteur et 3 fois en largeur`
- **Concaténation verticale**
  - `np.concatenate([arr,-arr,arr], axis=0)`
  - `np.vstack([arr,-arr,arr])`
- **Concaténation horizontale**
  - `np.concatenate([arr,-arr,arr], axis=1)`
  - `np.hstack([arr,-arr,arr])`
- **Découpage**
  - `c, d, e = np.vsplit(arr, 3) # découpage vertical (en parties équitables)`
  - `c, d, e = np.hsplit(arr, 3) # découpage horizontal (en parties ajustables)`