SEANCE 4

Le module NumPy

5 novembre 2025

Présentation

On commence par importer le module `Numpy` en début de programme.

```
import numpy as np
```

On a alors accès à une vaste bibliothèque de fonctions mathématiques.

Fonctions mathematiques usuelles

- `np.exp()`, `np.sin()`, `np.cos()`...
- `np.pi`
- `np.sign()`
- ...

Présentation

NumPy est un outil performant pour la manipulation de tableaux à plusieurs dimensions.

Il ajoute en effet le type `array`, qui est similaire à une liste, mais dont tous les éléments sont du même type : des complexes, des flottants, des entiers, ou des booléens.

Le module NumPy possède des fonctions basiques en algèbre linéaire, ainsi que pour les transformées de Fourier.

Création d'un tableau dont on connaît la taille

```
>>> import numpy as np
>>> a = np.zeros(4)
>>> a
array([ 0., 0., 0., 0.])


>>> nx = 2
>>> ny=2
>>> a=np.zeros((nx,ny))
>>> a
array( [ [ 0.,  0. ] ,
         [ 0.,  0.] ] )
```

Création d'un tableau dont on connaît la taille

Un tableau peut être multidimensionnel, comme ici, de dimension 3 :

```
>>> a=np.zeros((nx,ny,3))
>>> a
array( [ [ [ 0.,  0.,  0. ],
           [ 0.,  0.,  0. ] ],

         [ [ 0.,  0.,  0. ],
           [ 0.,  0.,  0. ] ] ] )
```

Mais nous nous limiterons dans ce cours aux tableaux uni et bi-dimensionnels.

Création d'un tableau dont on connaît la taille

Il existe également les fonctions `np.ones`, `np.eye`, `np.identity`, `np.empty`, ...

Par exemple :

- `np.empty` crée un tableau vide,
- `np.ones(5)` crée le tableau à une ligne `[1 1 1 1 1]`,
- `np.eye(3,2)` crée le tableau à 3 lignes et 2 colonnes contenant des 1 sur la diagonale et des zéros partout ailleurs.

Par défaut les éléments d'un tableaux sont des `float` (un réel en double précision) ; mais on peut donner un deuxième argument qui précise le type (`int`, `complex`, `bool`, ...).

Exemple :

```
>>> np.eye(2, dtype=int)
```

III - Création d'un tableau avec une séquence de nombres

Les fonctions `np.linspace` et `np.arange` sont très utiles calculer des fonctions puis tracer leurs graphes (cf séance 4) !

```
>>> a = np.linspace(-4, 4, 9)
>>> a
array([-4., -3., -2., -1., 0., 1., 2., 3., 4.])

>>> a = np.arange(-4, 4, 1)
>>> a
array([-4, -3, -2, -1, 0, 1, 2, 3])

>>> a = np.array([1, 2, 3])
>>> a
array([1, 2, 3])
```

Création d'un tableau à partir d'autres objets

```
>>> b = np.array(range(10))
>>> b
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> L1 = [1, 2, 3]
>>> L2 = [4, 5, 6]
>>> a = np.array([L1, L2])
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
```

Manipulations d'un tableau

- **a.shape** : retourne les dimensions du tableau
- **a.dtype** : retourne le type des éléments du tableau
- **a.size** : retourne le nombre total d'éléments du tableau
- **a.ndim** : retourne la dimension du tableau (1 pour un vecteur, 2 pour une matrice)

2. Indexation d'un tableau

Comme pour les listes et les chaines de caractères, l'indexation d'un vecteur (ou tableau de dimension 1) commence à 0. Pour les matrices (ou tableaux de dimension 2), le premier index se réfère à la ligne, le deuxième à la colonne.

Quelques exemples

```
>>> L1, L2 = [1, -2, 3], [-4, 5, 6]
>>> a = np.array([L1, L2])
>>> a[1, 2]
6

>>> a[:, 1]
array([-2, 5])

>>> a[1,0:2:2]
array([-4])

>>> a[:, -1:0:-1]
array([[3, -2],
[6, 5]])

>>> a[a < 0]
array([-2, -4])
```

Redimensionnement d'un tableau

Fonctions a.shape et a.reshape

```
>>> a = np.linspace(1, 6 ,6)
>>> a
array([  1.,   2.,   3.,   4.,   5.,   6.])

>>> a.shape = (2, 3)
>>> a
array([[ 1., 2., 3.],
[ 4.,5.,6.,]])

>>> a.shape = (a.size,)
>>> a
array([  1.,   2.,   3.,   4.,   5.,   6.])

>>> a.reshape(2, 3)
array([[ 1., 2., 3.],
[ 4.,5.,6.,]])

>>> a
array([  1.,   2.,   3.,   4.,   5.,   6.])
```

Boucles sur les tableaux

```
>>> a = np.zeros((2, 3))
>>> for i in range(a.shape[0]):
...     for j in range(a.shape[1]):
...         a[i, j] = (i + 1)*(j + 1)
>>> print a
[[ 1.  2.  3.]
 [ 2.  4.  6.]]


>>> for e in a:
... print e
[ 1.  2.  3.]
[ 2.  4.  6.]
```

Calculs sur les tableaux

Lorsqu'on utilise NumPy, il faut **éviter d'utiliser des boucles** !

Faire directement les calculs sur des tableaux !

Calculs sur les tableaux

Comparons l'exécution de deux bouts de programme où l'on calcule de deux façon différentes $y = 3x - 1$ pour un million de valeurs de x :

```
import numpy as np
import time

x=np.linspace(0,1,int(1e+6))
tic = time.time()
y=3*x-1
toc=time.time()
print(toc -tic)

tic = time.time()
y = np.zeros(int(1e+6))
for i in range(x.size):
    y[i] = 3*x[i] - 1
toc = time.time()
print (toc -tic)
```

renvoie

```
0.005846500396728516
1.538684129714965 8
```

Opérations terme à terme

L'opération c=a*b correspond à la multiplication terme à terme des éléments des tableaux a et b : $c[i,j] = a[i,j] * b[i,j]$.
De même, l'opération `A**2` correspond à une élévation au carré terme à terme.

Remarque pour plus tard : les tableaux à deux dimensions peuvent être aussi assimilés à des **matrices**.
Le produit matriciel se fait avec la fonction `dot()`.

Attention, si le deuxième argument est un vecteur ligne, il sera transformé si besoin en vecteur colonne, par transposition.

Ceci est du au fait qu'il est plus simple de définir un vecteur ligne, par exemple `v=np.array([0,1,2])`, qu'un vecteur colonne, ici `v=np.array([[0],[1],[2]])`.

Entrées et Sorties

Numpy peut lire – `numpy.loadtxt()` – ou sauvegarder – `numpy.savetxt()` – des tableaux dans un simple fichier ASCII :

```
>>> x = np.linspace(-1, 1, 100)

>>> # Sauvegarde dans le fichier 'archive_x.dat'
>>> np.savetxt('donnees.dat', x)

>>> # Relecture a partir du fichier 'archive_x.dat'
>>> y = np.loadtxt('donnees.dat')

# Test d'egalite stricte
>>> (x == y).all()
True
```