

Institut Préparatoire aux Études d'Ingénieurs de Tunis



Leila Ben Othman

Année universitaire: 2016 - 2017

Les tableaux en numpy

Plan

- ➊ Introduction
- ➋ Les tableaux unidimensionnels
- ➌ Les tableaux bidimensionnels
- ➍ Création de tableaux particuliers
- ➎ Opérations arithmétiques et fonctions mathématiques sur les tableaux
- ➏ Méthodes usuelles
- ➐ Algèbre linéaire avec `numpy`

Introduction

- Le module `numpy` est un module PYTHON spécialisé dans la manipulation des tableaux à n dimensions (`ndarray`). Dans le cadre de notre cours, on s'intéressera aux tableaux à 1 et à 2 dimensions.
- Le module `numpy` est indispensable pour faire du calcul scientifique avec PYTHON. Tous les autres modules tels que `scipy` et `matplotlib` se base sur le type `ndarray` de `numpy`.
- Les tableaux `numpy` sont homogènes, c'est-à-dire constitués d'éléments de même type.

NB:

- Site officiel de `numpy`: <http://www.numpy.org/>
- Le module `numpy` est disponible à l'adresse suivante:
<https://sourceforge.net/projects/numpy/files/>

Création d'un tableau (1)

- Import du module `numpy`

```
import numpy as np
```

- Création d'un tableau à une dimension avec la fonction `array`
(Typage implicite)

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

- L'objet créé par la fonction `array` est de type `ndarray`.

```
> type(T)  
<class 'numpy.ndarray'>
```

Création d'un tableau (2)

- Tous les éléments d'un tableau sont de même type. L'attribut dtype (data type) permet de voir le type des éléments.

```
> T.dtype  
dtype('int32')
```

- Création d'un tableau avec un typage explicite en utilisant dtype

```
> T=np.array([1,2,3],dtype=float)  
> T  
array([ 1., 2., 3.])  
> T.dtype  
dtype('float64')
```

- Les principaux types sont : bool, int, float, complex

Taille - Accès à un élément du tableau

- On calcule la taille d'un tableau T (le nombre d'éléments) à l'aide la fonction `len` (`len(T)`);
- On accède à chaque élément du tableau par son index:
 - $T[0]$: premier élément de T
 - $T[\text{len}(T) - 1]$: dernier élément de T
- Lorsque l'index est négatif, le parcours est effectué à rebours à partir de la fin:
 - $T[-1]$: premier élément de T (de droite vers la gauche).
 - $T[-\text{len}(T)]$: dernier élément de T (de droite vers la gauche).

Attributs d'un tableau

- Le nombre de dimensions

```
> T.ndim  
1
```

- Le nombre d'éléments

```
> T.size  
3
```

- Le nombre d'éléments sur chaque dimension

```
> T.shape  
(3,) # tuple: 3 éléments sur la première dimension
```

Attributs d'un tableau

- **Attention:** Les attributs `ndim`, `size` et `shape` peuvent être évoquées à la fois comme des attributs (caractéristiques) d'un tableau et comme des fonctions du module `numpy` prenant un tableau en paramètre.
- Par exemple pour l'attribut `shape`, on peut donc écrire aussi bien `T.shape` que `np.shape(T)`.
- En deuxième année, on s'intéressera à la signification exacte de la notion d'attribut, qui est en rapport avec le fait que PYTHON est un langage orienté objet. Encore un peu de patience :)

Découpage ou Slicing sur un tableau

- Comme pour les séquences, le module `numpy` permet le découpage du tableau en tranches avec la syntaxe suivante $T[debut : fin]$
- $T[debut : fin]$ désigne la tranche de T constituée des éléments dont les index sont compris entre *debut* et *fin* - 1.
- Quand *debut* n'est spécifié, alors la tranche commencera du début de T
- Quand *fin* n'est spécifié, alors la tranche finira à la fin de T
- La syntaxe $T[debut : fin : pas]$ possède un troisième paramètre (par défaut, égal à 1), indiquant le pas de sélection.

Création d'une matrice (1)

- Création d'un tableau à deux dimensions (matrice) avec la fonction `array`

```
> M=np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]])  
>M  
array([[ 1, 2, 3],  
       [ 4, 5, 6],  
       [ 7, 8, 9],  
       [10, 11, 12]])
```

- L'objet crée par la fonction `array` est toujours de type `ndarray`.

```
> type(M)  
<class 'numpy.ndarray'>
```

Taille - Accès à un élément/une ligne d'une matrice

- La fonction `len(M)` retourne le nombre de lignes de M .
- On accède à chaque élément de la matrice par son index ligne et index colonne:
 - $M[i, j]$: élément qui se trouve à la ligne i et à la colonne j .
- On peut récupérer toute la ligne i avec la syntaxe $M[i]$. Le résultat est un tableau à une dimension.
 - $M[0]$: première ligne de M .
 - $M[\text{len}(M)-1]$: dernière ligne de M .
- Lorsque l'index est négatif, le parcours est effectué à rebours à partir de la fin:
 - $M[-1]$: dernière ligne de M .
 - $M[-\text{len}(M)]$: première ligne de M .

Taille - Accès à un élément/une ligne d'une matrice

```
> len(M)  
4
```

```
> M[0,2]  
3
```

```
> M[2]  
array([7, 8, 9])
```

```
> M[0]  
array([1, 2, 3])
```

```
> M[-len(M)]  
array([1, 2, 3])
```

```
> M[-1]  
array([10, 11, 12])
```

```
> M[len(M)-1]  
array([10, 11, 12])
```

```
> M[-1,-1]  
12
```

Attributs de la matrice

- Le nombre de dimensions

```
> M.ndim  
2
```

- Le nombre d'éléments

```
> M.size  
6
```

- Le nombre d'éléments sur chaque dimension: nombre de lignes, nombres de colonnes

```
> M.shape  
(2,3) # tuple: 2 lignes et 3 colonnes
```

Découpage ou Slicing sur une matrice

```
>M=np.array([[1,2,3,4,5,6,7,8],[9,10,11,12,14,15,16,17],  
             [18,19,20,21,22,23,24,25],[26,27,28,29,30,31,32,33]])  
> M  
array([[ 1, 2, 3, 4, 5, 6, 7, 8],  
       [ 9, 10, 11, 12, 14, 15, 16, 17],  
       [18, 19, 20, 21, 22, 23, 24, 25],  
       [26, 27, 28, 29, 30, 31, 32, 33]])
```

Lignes 1 à 3, colonnes 2 à 5

```
> M[1:4,2:6]  
array([[11, 12, 14, 15],  
       [20, 21, 22, 23],  
       [28, 29, 30, 31]])
```

Lignes et colonnes paires

```
> M[:,::2]  
array([[ 1, 3, 5, 7],  
       [18, 20, 22, 24]])
```

Découpage ou Slicing sur une matrice

Lignes et colonnes impaires

```
> M[1::2,1::2]  
array([[10, 12, 15, 17],  
       [27, 29, 31, 33]])
```

Trois premières lignes, deux premières colonnes

```
> M[:3,:2]  
array([[ 1,  2],  
       [ 9, 10],  
       [18, 19]])
```

Découpage ou Slicing sur une matrice

à partir de la ligne 2, à partir de la colonne 4

```
> M[2:,4:]  
array([[22, 23, 24, 25],  
       [30, 31, 32, 33]])
```

on inverse l'ordre des colonnes

```
> M[:,::-1]  
array([[ 8, 7, 6, 5, 4, 3, 2, 1],  
       [17, 16, 15, 14, 12, 11, 10, 9],  
       [25, 24, 23, 22, 21, 20, 19, 18],  
       [33, 32, 31, 30, 29, 28, 27, 26]])
```

Découpage ou Slicing sur une matrice

on inverse l'ordre des lignes

```
> M[::-1, :]
array([[26, 27, 28, 29, 30, 31, 32, 33],
       [18, 19, 20, 21, 22, 23, 24, 25],
       [ 9, 10, 11, 12, 14, 15, 16, 17],
       [ 1, 2, 3, 4, 5, 6, 7, 8]])
```

on inverse l'ordre des lignes et l'ordre des colonnes

```
> M[::-1, ::-1]
array([[33, 32, 31, 30, 29, 28, 27, 26],
       [25, 24, 23, 22, 21, 20, 19, 18],
       [17, 16, 15, 14, 12, 11, 10, 9],
       [ 8, 7, 6, 5, 4, 3, 2, 1]])
```

Accès à une colonne

- La syntaxe $M[:,b]$ permet d'accéder à la colonne d'indice b . Le résultat est un tableau à une dimension.
- De même, la syntaxe $M[:,b:b+1]$ permet d'accéder à la colonne d'indice b . Le résultat est un tableau à deux dimensions.

Accès à la colonne d'indice 3

```
> M[:,3]  
array([ 4, 12, 21, 29])
```

Accès à la colonne d'indice 3

```
> M[:,3:4]  
array([[ 4],  
       [12],  
       [21],  
       [29]])
```

Création de tableaux particuliers

- Tableau d'une dimension de 5 valeurs aléatoires (par défaut, des réels)

```
> np.empty(5)
array([ 2.79206344e+178, 2.25315555e-316, 0.00000000e+000,
        2.80858511e+178, 6.35862486e-321])
```

```
> np.ndarray(5)
array([ 2.79206344e+178, 2.25316108e-316, 0.00000000e+000,
        2.80858511e+178, 6.35862486e-321])
```

Création de tableaux particuliers

- Tableau d'une dimension de 5 valeurs égales à 1.

```
> np.ones(5)
array([ 1.,  1.,  1.,  1.,  1.])
> np.ones(5,int)
array([1, 1, 1, 1, 1])
```

- Tableau d'une dimension de 5 valeurs égales à 0.

```
> np.zeros(5)
array([ 0.,  0.,  0.,  0.,  0.])
```

Création de tableaux particuliers

- La fonction `arange( $d, f, h$ )` permet de créer un tableau de valeurs de l'intervalle $[d, f[$ avec un pas de h .
- Les paramètres d et h sont optionnels.

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

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

```
> np.arange(1,10,2)
array([1, 3, 5, 7, 9])
```

Création de tableaux particuliers

- La fonction `linspace(a, b, n)` permet de créer un tableau de n valeurs régulièrement échelonnées de l'intervalle $[a, b]$ (les bornes de l'intervalle sont incluses). Par défaut, n est égale à 50.

```
> np.linspace(0,100,4)
array([ 0. , 33.33333333, 66.66666667, 100. ])
```

pour exclure la borne supérieure de l'intervalle:

```
> np.linspace(0,100,4,endpoint=False) array([ 0., 25., 50., 75.])
```

pour connaître la valeur du pas utilisé pour générer les éléments:

```
> np.linspace(0,100,4,retstep=True) (array([ 0. , 33.33333333, 66.66666667, 100. ]), 33.33333333333336)
```

Création de matrices particulières

- Tableau de deux dimensions de valeurs aléatoires (2 lignes, 2 colonnes).

```
> np.empty((2,2))  
array([[ 9.61395921e-293,  9.26242852e-293],  
       [ 9.08574574e-293,  6.35862486e-321]])
```

```
> np.ndarray((2,2))  
array([[ 9.81579501e-293,  9.81564210e-293],  
       [ 9.09881919e-293,  6.35862486e-321]])
```

Création de matrices particulières

- Tableau de deux dimensions dont les valeurs sont égales à 1 (2 lignes, 2 colonnes).

```
> np.ones((2,2))  
array([[ 1.,  1.],  
       [ 1.,  1.]])
```

- Tableau de deux dimensions dont les valeurs sont égales à 0 (2 lignes, 2 colonnes).

```
> np.zeros((2,2))  
array([[ 0.,  0.],  
       [ 0.,  0.]])
```

Création de matrices particulières

- La fonction *identity*(n) permet de créer la matrice identité d'ordre n .

```
> np.identity(4)
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

Création de matrices particulières

- La fonction `diag`, permet de créer une matrice diagonale formée par les coefficients passés en paramètres (sous forme de liste, de tuple ou même de tableau).

```
> np.diag([1,2,3,4])  
array([[1, 0, 0, 0],  
       [0, 2, 0, 0],  
       [0, 0, 3, 0],  
       [0, 0, 0, 4]])
```

```
> np.diag((1,2,3,4))  
array([[1, 0, 0, 0],  
       [0, 2, 0, 0],  
       [0, 0, 3, 0],  
       [0, 0, 0, 4]])
```

Opérations arithmétiques

Soient A et B deux tableaux `numpy`, les opérations mathématiques suivantes sont disponibles:

- Addition: $A + B$ additionne terme à terme les éléments de A et B
- Soustraction: $A - B$ soustrait terme à terme les éléments de B à ceux de A
- Multiplication: $A * B$ multiplie terme à terme les éléments de A et B
- Division: A / B quotients terme à terme des éléments de A par ceux de B
- Puissance: $A ** B$ élève les éléments de A à la puissance les éléments de B

NB: Toutes ces opérations sont également applicables avec un scalaire: addition des éléments d'un tableau et un scalaire, multiplication des éléments par un scalaire, *etc.*

Fonctions mathématiques

- Toutes les fonctions mathématiques usuelles sont disponibles dans le module `numpy` et restent applicables sur des tableaux à une ou deux dimensions. Citons par exemple: les fonctions `sin`, `cos`, `tan`, `log`, `sqrt`, `exp`, *etc.*

```
> T=np.linspace(-3*np.pi,3*np.pi,5)
> T
array([-9.42477796, -4.71238898, 0. , 4.71238898, 9.42477796])

> np.sin(T)
array([-3.67394040e-16, 1.00000000e+00,
 0.00000000e+00,-1.00000000e+00, 3.67394040e-16])
```

Méthodes usuelles

- `M.sum()` renvoie la somme des éléments de `M`.
- `M.min()` renvoie le minimum des éléments de `M`.
- `M.max()` renvoie le maximum es éléments de `M`.

NB:

- Quand le paramètre `axis=0` est spécifié, la méthode `M.sum(axis=0)` (resp `M.min(axis=0)`, `M.max(axis=0)`) retourne un tableau composé respectivement de la somme, du min et du max des éléments de chaque colonne.
- Quand le paramètre `axis=1` est spécifié, la méthode `M.sum(axis=1)` (resp `M.min(axis=1)`, `M.max(axis=1)`) retourne un tableau composé respectivement de la somme, du min et du max des éléments de chaque ligne.

Méthodes usuelles

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

```
> M
```

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

```
> M.sum()
```

```
21
```

```
> M.sum(axis=0)
```

```
array([5, 7, 9], dtype=int32)
```

```
> M.sum(axis=1)
```

```
array([ 6, 15], dtype=int32)
```

Produit scalaire et transposé

- La fonction produit scalaire est disponible directement sous numpy sous le nom `dot`. De même pour la transposition de matrice (`transpose`).

```
> A = np.array([[1,2],[3,4]])
> B = np.array([[2,0],[3,4]])
> np.dot(A,B)
array([[ 8,  8],
       [18, 16]])

> A
array([[1, 2],
       [3, 4]])
> np.transpose(A)
array([[1, 3],
       [2, 4]])
```

Le module linalg

- Afin d'utiliser d'autres méthodes d'algèbre linéaire, il faut utiliser le sous module `linalg` de `numpy` qu'il faudra importer avec la commande: `import numpy.linalg as lng`

Déterminant `lng.det(M)`

Inverse `lng.inv(M)`

Vecteurs propres `lng.eig(M)`

Valeurs propres `lng.eigvals(M)`

Résolution d'un système linéaire `lng.solve(A,B)` : retourne le vecteur X
solution du système $A \cdot X = B$