Informatique 5
matplotlib, numpy arrays

Rémi Emonet – 2017
Université Jean Monnet – Laboratoire Hubert Curien

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

« TD » de Lundi

- Contrôle 1
  - Lundi 9 Octobre, amphi D201
  - Contenu parmi
    - bases de Python (variables, boucles, listes, conversion)
    - fonctions en Python (définition, appel)
    - démarche d'écriture d'algorithme

Informatique 5 : Plan

- Logistique : rappels
- Bibliothèque Numpy : premier contact
- Numpy : type, *shape*, agrégations
- Numpy : opérations élément par élément
- Listes Python : tranches
- Numpy : indices et tranches
- Bibliothèque Matplotlib : premier contact

Numpy

- Motivation
  - `les_carrés`, `les_logs`, ...
  - `liste_zeros2d`, `grille`, ...
  - simplifié grâce aux « listes en compréhension » ⇒ peut mieux faire
  - + performances (vitesse de calcul)
- Bibliothèque Numpy (module `numpy`)
  - structure de données pour des tableaux à 1, 2, n dimensions
  - simplification des calculs
  - fonctions utilitaire pour les opérations classiques
  - bases de nombreuses autres bibliothèques

Numpy, ex. : création de tableaux de 0

- Exemple (atypique, préférer la version d'en bas)

```
1 import numpy
2 a = numpy.zeros(6)       # "vecteur" avec 6 zéros
3 print(a)
4
5
6 b = numpy.zeros([5, 10]) # tableau de 5 lignes et 10 colonnes
7 print(b)
```

- Version typique (`np` et utilisation de « tuple » (n-uplet))

```
1 import numpy as np
2 a = np.zeros(6)
3 print(a)
4
5 b = np.zeros((5, 10))
6 print(b)
```

- NB: `(5, 10)` est un **n-uplet (ou *tuple*)** qui se comporte comme la liste `[5, 10]` mais n'est pas modifiable (erreur si on essaie de changer une valeur)

Numpy : quelques fonctions de création

- `numpy.zeros(n)`
- `numpy.zeros(shape)` (*shape* ⇔ forme ⇔ dimensions du tableau)
- `np.eye(n)`
- `np.eye(n1, n2)`
- `np.arange(n)`
- `np.arange(a, b)`
- `np.arange(a, b, s)`
- `np.random.random(n)`
- `np.random.random(shape)`
- `np.random.uniform(a, b, shape)`
- `np.random.normal(moyenne, ecart_type, shape)`
- ...

Numpy : opérations

- La plupart des opération (`+`, `-`, `*`, ...) sont effectuée élément par élément
  - entre deux tableaux numpy `a + b`
  - entre un tableaux et un nombre `a + 10` (ajoute 10 à chaque élément de `a`)
- La plupart des fonctions de `math` ont un équivalent dans numpy
  - ex : `np.log(a)`
- Numpy propose des fonction d'agrégation
  - ex : `np.sum(a)`, `np.mean(a)`, `np.median(a)`, etc.

```python
import numpy as np

a = np.random.uniform(0.5, 1, (7, 4))
b = np.random.uniform(10, 20, (7, 4))

print(a+b)
print(a+10)
print(a**b)

print(np.sum(a))
print(np.sum(a, axis=0))
```


Informatique 5 : Plan

- Logistique : rappels
- Bibliothèque Numpy : premier contact
- Numpy : type, *shape*, agrégations
- Numpy : opérations élément par élément
- Listes Python : tranches
- Numpy : indices et tranches
- Bibliothèque Matplotlib : premier contact

Numpy : création et *shape* (forme, dimensions)

```python
import numpy as np
a = np.ones((3, 8))
b = np.arange(0, 100, 5)
c = np.arange(0.5, 100, 5)
d = np.array([10, 15, 3.14, 2.7, 9.81])
```

- Tableau numpy : type `ndarray` et type des éléments
  - `ndarray` tableau à n dimensions
  - types numériques python : int64, float64, bool, ...

```python
print( type(a), type(b), type(c) )  # numpy.ndarray (les trois)
print( a.dtype, b.dtype, c.dtype )  # float64 int64 float64
```

- *shape* ⇔ forme ⇔ dimensions de tableau numpy

```python
print(a.shape)  # (3,8)   2-uplet (paire) avec 3 et 8
print(b.shape)  # (20,)   1-uplet contenant la valeur 20
print(c.shape)  # (20,)
```

- Indice multiples (autant que de dimensions)

```python
print(a[0, 0])
print(b[0])
a[1, 1] = a[2, 2] + 999
```


Numpy : *reshape* et données partagées et *copy*

- Changement de forme avec `.reshape(shape)`

```python
a = np.array([1, 2, 3, 10, 20, 30])
b = a.reshape((2, 3))  # réorganiser en 2 lignes de 3 el.
# équivalent, calcul auto. de la dimension manquante
b = a.reshape((2, -1))
b = a.reshape((-1, 3))
```

- ⚠ Les tableaux et listes partagent leurs données

```python
print(a[0])  # 1
b[0,0] = 3.14
print(a[0])  # 3.14

l = [1, 2 ,3]  # idem en numpy sans reshape ou avec listes
# idem si : l = np.array([1, 2, 3])
v = l
v[0] = 3.14
print(l[0])  # 3.14
```

- Copier un tableau avec `.copy()`

```python
a[0] = 1
c = a.copy()
c[0] = 3.14
print(a[0])  # 1
```


Numpy : fonctions d'agrégations

```python
a = np.random.uniform(0.95, 1.05, (20, 50))

print(a.shape)   # (20, 50)
print(a.size)    # 1000
```

```python
print(a.sum())   # ≈ 1000
```

```python
print(a.mean())  # ≈ 1
print(a.var())   # ≈ 0.0008333 (0.1² / 12)
print(a.std())   # = a.var() ** 0.5
```

```python
print(a.min())   # ≥ 0.95
print(a.max())   # ≤ 1.05
```

- Indice du min/max

```python
b = np.array([[10, 20, 30],
              [60, 50, 40]])

print(b.argmin())          # 0
bligne = b.reshape((-1,))
print(bligne.argmax())     # 3

print(b.argmax())          # 3 (comme si en ligne)

print(np.unravel_index(b.argmax(), b.shape))   # (1, 0)
```


Informatique 5 : Plan

- Logistique : rappels
- Bibliothèque Numpy : premier contact
- Numpy : type, *shape*, agrégations
- Numpy : opérations élément par élément
- Listes Python : tranches
- Numpy : indices et tranches
- Bibliothèque Matplotlib : premier contact

Opérations élément par élément

- Créons deux tableaux de même dimensions

```
a = np.arange(0, 1000, 100).reshape((2, 5))
b = np.arange(0, 10).reshape((2, 5))

assert_allclose(a, np.array([[  0, 100, 200, 300, 400],
                             [500, 600, 700, 800, 900]]))

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

- Opérations éléments par élément

```
c = a+b
assert_allclose(c, np.array([[  0, 101, 202, 303, 404],
                             [505, 606, 707, 808, 909]]))
d = a*b
assert_allclose(d, np.array([[  0, 100, 400, 900,1600],
                            [2500,3600,4900,6400,8100]]))
```

```
print(a/b)
#   [[  nan  100.  100.  100.  100.]
#    [ 100.  100.  100.  100.  100.]]
# nan : « not a number » (valeur spéciale, « pas un nombre »)
```


Opérations entre un tableau et un nombre

- Créons un tableaux

```
a = np.arange(0, 1000, 100).reshape((2, 5))
assert_allclose(a, np.array([[  0, 100, 200, 300, 400],
                             [500, 600, 700, 800, 900]]))
```

- Opérations avec un nombre

```
print(a+1)                    # ou 1+a
#   [[  1 101 201 301 401]
#    [501 601 701 801 901]]

print(a/100)                  # ou a*0.01 ou 0.01*a
#   [[ 0.  1.  2.  3.  4.]
#    [ 5.  6.  7.  8.  9.]]

print(a**2)
#   [[     0  10000  40000  90000 160000]
#    [250000 360000 490000 640000 810000]]

print(100/a)
#   [[ inf         1.      0.5         0.3333  0.25  ]
#    [ 0.2         0.1667  0.14285714  0.125   0.1111]]
#
# inf : « infinity » (infini)
```


Fonctions mathématiques sur chaque élément

- Créons un tableaux

```
a = np.arange(0, 600, 100).reshape((2, 3))

assert_allclose(a, np.array([[  0, 100, 200],
                             [300, 400, 500]]))
```

- Fonctions mathématiques du module `numpy` ...

```
print(np.sin(a))
#   [[ 0.         -0.50636564 -0.8732973 ]
#    [-0.99975584 -0.85091936 -0.46777181]]
print(np.cos(np.radians(a)))
#   [[ 1.         -0.17364818 -0.93969262]
#    [ 0.5         0.76604444 -0.76604444]]
print(np.log(a))
#   [[      -inf  4.60517019  5.29831737]
#    [ 5.70378247  5.99146455  6.2146081 ]]
print(np.exp(a))
#   [[  1.00000000e+000   2.68811714e+043   7.22597377e+086]
#    [  1.94242640e+130   5.22146969e+173   1.40359222e+217]]
print(np.clip(a, 150, 350))
#   [[150 150 200]
#    [300 350 350]]
...
# https://docs.scipy.org/doc/numpy/reference/routines.math.html
# (^ pas que des opération élément par élément)
```


Informatique 5 : Plan

- Logistique : rappels
- Bibliothèque Numpy : premier contact
- Numpy : type, *shape*, agrégations
- Numpy : opérations élément par élément
- Listes Python : tranches
- Numpy : indices et tranches
- Bibliothèque Matplotlib : premier contact

Tranches de listes Python

- Rappel : accès à un élément d'une liste `la_liste[l_indice]`

```
a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
print(a[3])  # 30
```

- Extraction d'une sous partie de liste

```
b = a[3:6]
print(b)  # [30, 40, 50]

# avec un pas de 2
c = a[3:6:2]
print(c)  # [30, 50]
```

- Équivalent à

```
b = [a[i] for i in range(3, 6)]

c = [a[i] for i in range(3, 6, 2)]

# rappel : équivalent à
c = []
for in range(3, 6, 2):
    c.append(a[i])
```


Tranches en Python : raccourcis

- Omission des indices

```
a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

- Début et fin automatique

```
print(a[:3])     # [0, 10, 20]
print(a[5:])     # [50, 60, 70, 80, 90]
```

- Avec un pas

```
print(a[:5:2])   # [0, 20, 40]
print(a[5::2])   # [50, 70, 90]

# et que le pas
print(a[::3])    # [0, 30, 60, 90]
```

- Astuces

```
print(a[:])      # une copie de a

# pas négatif
print(a[::-1])  # [90, 80, 70, 60, 50, 40, 30, 20, 10, 0]
print(a[::-3])  # [90, 60, 30, 0]
```

Indices et tranches Python : indices négatifs

```
1  a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

- Indices à partir de la fin

```
1  print(a[-1])          # 90
2  print(a[len(a)-1])    # 90
3
4  print(a[-3])          # 70
5  print(a[len(a)-3])    # 70
```

- Marche aussi en tranches

```
1  print(a[1:-1])        # 10 ... 80
2  print(a[1:len(a)-1])  # 10 ... 80
3
4  print(a[:-3])         # 0 ... 60
```


Informatique 5 : Plan

- Logistique : rappels
- Bibliothèque Numpy : premier contact
- Numpy : type, *shape*, agrégations
- Numpy : opérations élément par élément
- Listes Python : tranches
- Numpy : indices et tranches
- Bibliothèque Matplotlib : premier contact

Numpy : indices et tranches

- Autant d'indices que de dimensions
- Pour chaque dimension, tout marche comme avec les liste
  - indices
  - indices négatif
  - tranches
    - valeurs par défaut
    - indices négatifs
    - pas
    - pas négatif

Re: Illustration de Numpy : π

Informatique 5 : Plan

- Logistique : rappels
- Bibliothèque Numpy : premier contact
- Numpy : type, *shape*, agrégations
- Numpy : opérations élément par élément
- Listes Python : tranches
- Numpy : indices et tranches
- Bibliothèque Matplotlib : premier contact

Matplotlib

- Motivation
  - tracer des graphes (courbes, nuages de points, histogrammes, etc)
  - gérer automatiquement l'affichage des axes, titres, graduations, etc.
  - permettre éventuellement l'interaction l'utilisateur (zoom, etc.)
  - permettre éventuellement de sauver des fichiers (jpg, pdf, etc.)
- Bibliothèque Matplotlib (module `matplotlib`)
  - généralement `from matplotlib import pyplot as plt`
  - tracé de différents types de graphes
  - gestion des sous graphes
  - http://matplotlib.org/
  - http://matplotlib.org/gallery.html

Matplotlib : exemple

```
from matplotlib import pyplot as plt

x = [1, 2, 3, 9, 10]
y = [ e**3 for e in x ]
z = [ (10+e)**2 for e in x ]

plt.plot(x, y, label="courbe 1")
plt.plot(x, z, label="courbe 2")
plt.legend()

plt.savefig('courbes.pdf')
plt.show()

```


Illustration de Numpy + Matplotlib : affichage des pas (quand on marche)