ENSAI - École nationale de la statistique et
de l'analyse de l'information, Rennes

Numpy, Matplotlib et Scipy en 10 minutes

Farah AIT SALAHT
farah.ait-salaht@ensai.fr

janvier 2017

Python pour le calcul scientifique et l'analyse de données

Python devient progressivement le langage le plus populaire pour l'analyse des données et le calcul scientifique, cependant des modules supplémentaires sont nécessaires. Parmi les modules les plus populaires (indispensables), on cite :

- Calcul scientifique

  NumPy manipulation de tableaux à N-dimension

  Matplotlib graphes 2D (similaire à Matlab)

  SciPy calcul scientifique (algèbre linéaire, intégration numérique, optimisation, etc.)

- Statistique et Science des Données

  Pandas structure de données et feuilles de calcul (fournit une structure de DataFrame similaire à R)

  Scikit-learn modélisation et algorithmes d'apprentissage statistique

Numpy

- Numpy est un ensemble de modules (packages) dédié au calcul scientifique en python
- Basé sur le type d'objet : tableau (**array**) multidimensionnel homogène. Il est associé à un type (entier, flottant, complexe ...).
- Des modules numpy fournissent des fonctions pour le calcul numérique, en particulier :
  - Algèbre Linéaire (numpy.linalg) : calcul matriciel, décompositions, résolution d'équations
  - Échantillonnage aléatoire (numpy.random) : Générateurs de nombres aléatoires, principales distributions, permutations, mélanges
- De loin, la structure de donnée la plus utilisée pour les maths numériques sous Python

Créer un tableau à N-dimension avec NumPy

- Pour importer le module `Numpy`, saisir la commande suivante :

```
import numpy as np
```

np est une abréviation d'import classique.

Il existe de nombreuses façons de créer des tableaux à N-dimensionnel

- `arange()` : génère un array (idem que `range` pour une liste)

```
np.arange(5)                # <--->    np.array([0, 1, 2, 3, 4])
np.arange(3, 7, 2)          # <--->    np.array([3, 5])
```

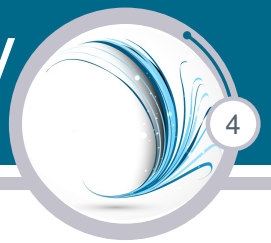
- `ones()`, `zeros()` et `eye()` :

```
np.ones(3) # np.array([1., 1., 1.])

np.ones((3,4)) #np.array([1., 1., 1.],[1., 1., 1.],[1., 1., 1.])

np.eye(3)  # np.array([1., 0., 0.],[0., 1., 0.],[0., 0., 1.])
```

- `linspace(star, stop, spacing)` : produit un vecteur avec des éléments espacés linéairement

```
np.linspace(3, 7, 3)          # <--->     np.array([3., 5., 7.]])
```

Créer un tableau à N-dimension avec NumPy



- Matrices aléatoires avec `from numpy.random import *` : `rand(size)`, `randn(size)`, `normal(mean,stdev,size)`, `uniform(low,high,size)`, `randint(low,high,size)`

```
N = int(1e7)
from random import normalvariate
%timeit [normalvariate(0,1) for i in range(N)]
```

10.9 s

```
N = int(1e7)
%timeit np.random.randn(N)
```

376 ms

- `genfromtxt()` : génère un tableau à partir d'un fichier

```
a = np.genfromtxt('data.csv', dtype=np.float64, delimiter=',')
    # Creer un tableau a partir d'un fichier CSV
```

Créer un tableau à N-dimension avec NumPy

Autres fonctions utiles pour les Arrays

```
a=np.array([[0,1],[2,3],[4,5]]) # Creer un tableau initialise
    par une liste de listes

np.ndim(a), a.ndim    --> 2    # retourne la dimension du tableau

np.size(a), a.size    --> 6    # Nombre d'elements du tableau

np.shape(a), a.shape -->(3, 2)#Tuple contenant la dimension de a

a.dtype               --> dtype('int32') #  type des donnees

np.transpose(a)    --> array([[0, 2, 4],[1, 3, 5]]) # Transpose

a.min(), np.min(a)--> 0                  # Valeur min

a.sum(), np.sum(a)--> 15                 # Somme des valeurs
a.sum(axis=0)      --> array([6, 9])     # Somme sur les colonnes
a.sum(axis=1)      --> array([1, 5, 9]) # Somme sur les lignes
```

- mais aussi max, mean, std, ...

Tous les éléments doivent avoir le même type (booléen, entier, réel, complexe)

Récupérer / Modifier les valeurs d'un tableau

- NumPy offre de nombreuses façons d'extraire des données à partir des tableaux

```
print(a[0,0])   # Afficher un seul element d'un tableau 2D

print(a[0,:])  # Print la premiere ligne d'un tableau 2D -> 1D array

print(a[:,-1]) # Print la derniere colonne d'un tableau -> 1D array

print(a[0:2,1:3]) # Afficher une sous-matrice d'un tableau 2D ->
    tableau 2D
```

- NumPy utilise la même syntaxe de base pour modifier les tableaux

```
a[0,0] = 25.0  # Affecter une valeur a un element d'un tableau 2D

a[0,:] = np.array([10,11,12], dtype=np.float64)
# Affecter un tableau 1D a la premiere ligne d'un tableau 2D

a[:,-1] = np.array([20,21], dtype=np.float64)
 # Affecter un tableau 1D a la derniere colonne d'un tableau 2D

a[0:2,1:3] = np.array([[10,11],[20,21]], dtype=np.float64)
# Affecter un tableau 2D a une sous-matrice d'un tableau 2D
```

Opérations sur les tableaux

- **Opérations sur les tableaux :** opérateurs binaires et des fonctions Numpy

```
In [ ]: a = np.arange(4, dtype=np.float64) # Creation d'un tableau 1D

In [ ]: a + a # addition de deux tableaux a 1D

In [ ]: a * a # Multiplication element par element de 2 tableaux 1D

In [ ]: a.sum()   # Sommer les elements d'un tableau 1D

In [ ]: np.dot(a, a) # ou (a * a).sum(), Calcul le produit scalaire

In [ ]: np.dot(a.reshape(4,1), a.reshape(1,4))   # Calcul du produit
    vectoriel
```

Changement de type

- **Changement de type :**

```
In [ ]: T = np.array([[-6, 2], [0, 1]])
In [ ]: print(T.dtype)
int32
```

```
In [ ]: T2 = T.astype(float)  # conversion en float
In [ ]: print(T2)
In [ ]: print(T2.dtype)
[[-1.  2.]
 [ 0.  4.]]
float64
```

```
In [ ]: T3 = T.astype(bool)  # conversion en bool
In [ ]: print(T3)
In [ ]: print(T3.dtype)
[[ True  True]
 [False  True]]
bool
```

Enregistrer un tableau dans un fichier

- A l'aide de numpy.savetxt on peut enregistrer un array numpy dans un fichier txt

```
In [ ]: from numpy import *
In [ ]: T = random.standard_normal((3, 3))
In [ ]: print(T)
[[ 0.81399963  1.83533954 -0.69508429]
 [ 1.77410623 -0.65021135 -1.98917916]
 [ 1.06127485 -0.55764509 -0.33679266]]

In [ ]: np.savetxt("random-data.csv", T,
            fmt=' % 2.3f', delimiter=',')
In [ ]: edit random-data.csv
  0.814,  1.835, -0.695
  1.774, -0.650, -1.989
  1.061, -0.558, -0.337
```

- **Format de fichier Numpy natif :** Pour sauvegarder et recharger des tableaux numpy : np.save et np.load

```
In [ ]: np.save("random-matrix.npy", T)

In [ ]: np.load("random-matrix.npy")
array([[ 0.81399963,  1.83533954, -0.69508429],
       [ 1.77410623, -0.65021135, -1.98917916],
       [ 1.06127485, -0.55764509, -0.33679266]])
```

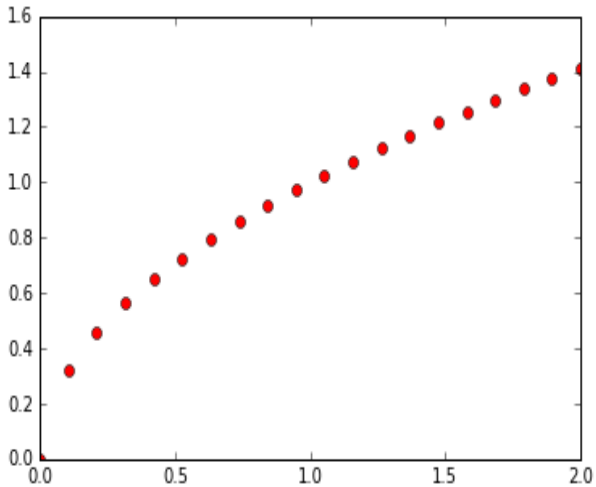
Matplotlib

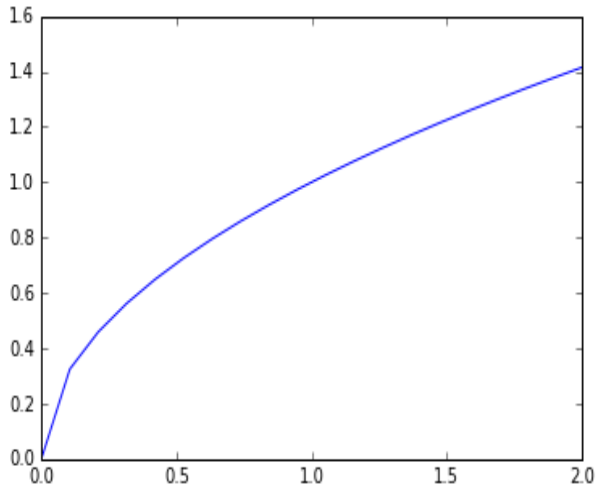
- Bibliothèque de représentation graphique (2D et 3D)
- Commandes proches de celles sous matlab
- Aussi connu sous le nom de pylab
- On appel la Matplotlib avec :
  - **import** pylab
    ou
  - **import** matplotlib.pyplot as plt
- Pour se faire une idée de ce qu'on peut faire avec Matplotlib, voir la page :

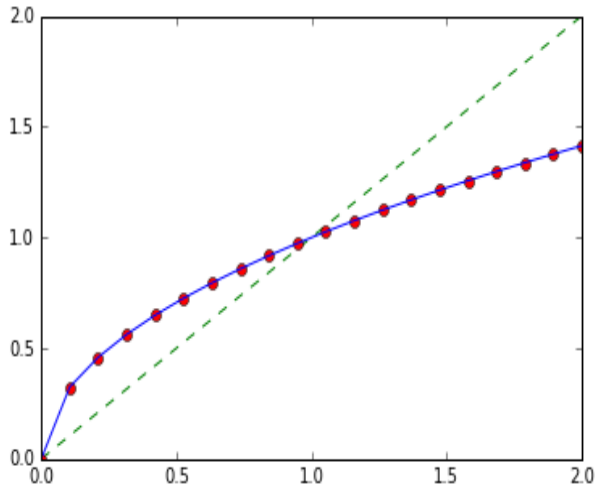
  `http://matplotlib.org/1.3.1/gallery.html`

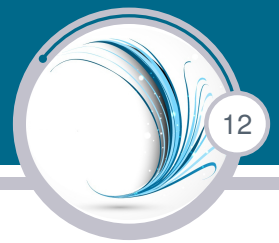
Graphiques 2D en utilisant Matplotlib


```
import numpy as np
import matplotlib.pyplot as plt     # or from pylab import *

x = np.linspace(0.0, 2.0, 20)

plt.plot(x, np.sqrt(x), 'ro')    # red circles
plt.show()

plt.plot(x, np.sqrt(x), 'b-')    # blue lines
plt.show()

# Three plots in one figure
plt.plot(x, x, 'g--', x, np.sqrt(x), 'ro', x, np.sqrt(x), 'b-')
plt.show()
```









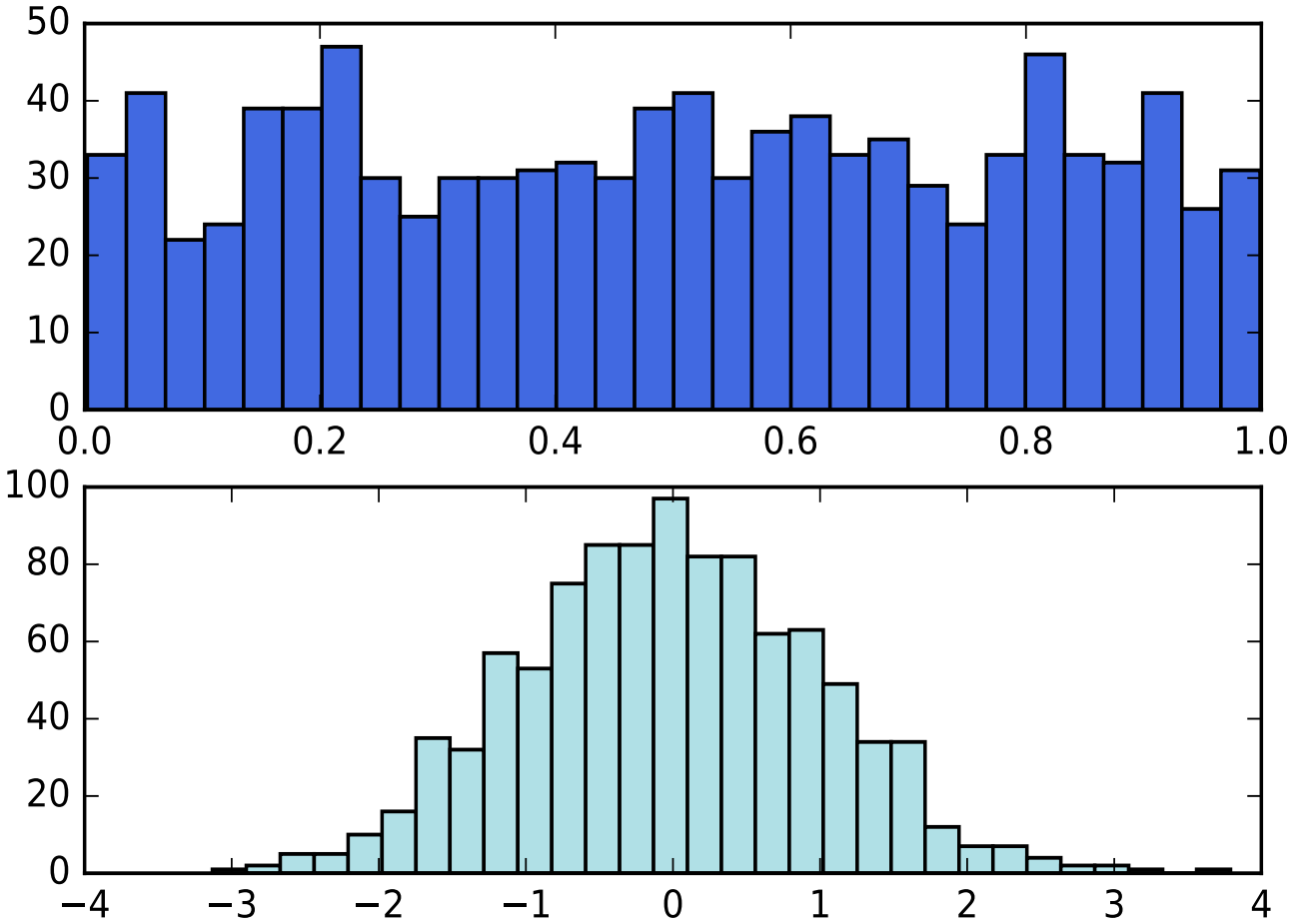
Tracé d'un histogramme

Plusieurs graphes sur une même figure :

```
import numpy as np
import matplotlib.pyplot as plt

y = np.random.rand(1000) # tirage aleatoire uniforme dans [0,1]
print(type(y))
print(y.shape)

plt.subplot(211)
plt.hist(y, 30, color="RoyalBlue")

y = np.random.randn(1000) # loi normale centree reduite
plt.subplot(212)
plt.hist(y, 30, color="PowderBlue")
plt.savefig('exemple2_matplot.pdf')
plt.show()
```

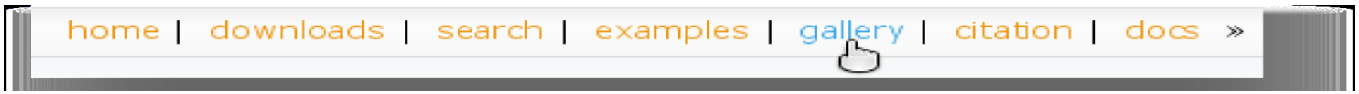
Graphiques 2D en utilisant Matplotlib

Tracé d'histogrammes



Graphiques 2D en utilisant Matplotlib

- Matplotlib possède une galerie impressionnante de tracés possibles : `http://matplotlib.org/gallery.html`



  - Tous les graphiques sont accompagnés du programme Python correspondant
  - Approche par l'exemple (copier/coller/modifier...)
- Toute la doc est sur le site `matplotlib.org/contents.html`

SciPy

Pour voir ce que l'on peut faire avec Scipy, consulter la page :

http://docs.scipy.org/doc/scipy/reference/

Bibliothèque Scipy :

- Représente ensemble très complet de modules d'algèbre linéaire, statistiques et autres algorithmes numériques
- Étend le panel de fonction de numpy (Quelques fonctions redondantes avec celles de NumPy)
- scipy.linalg = extension de numpy.linalg
- scipy.stats : accès à l'ensemble des distributions statistiques /tests ...

- Fonctions de SciPy plus évolués que celles de NumPy
- Celles de NumPy sont là pour assurer la compatibilité avec du vieux code

SciPy : fonctions d'algèbre linéaire


```
import numpy as np
from scipy import linalg

a = np.array([[1, 2], [3, 4]], dtype=np.float64)

# Compute the inverse matrix
linalg.inv(a)

# Compute singular value decomposition
linalg.svd(a)

# Compute eigenvalues
linalg.eigvals(a)
```

Exemple de fonction en statistique


```
import numpy as np
import scipy.stats as stat

d = np.array([0.553,0.57,0.576,0.601,0.606,0.606,0.609,0.611,0.615,
0.628,0.654,0.662,0.668,0.67,0.672,0.69,0.693,0.749])

#objet statistique descriptives
In []: stat_des = stat.describe(d)
In []: print(stat_des)

DescribeResult(nobs=18, minmax=(0.5530000000000005, 0.749),
     mean=0.6351666666666666, variance=0.002536852941176471,
     skewness=0.3876328997975213 6, kurtosis=-0.358736904875191 6)

#mediane de NumPy
print(np.median(d)) # 0.6215

#fonction de repartition empirique
print(stat.percentileofscore(d,0.6215))
 50.0   # la moitie des obs. ont une valeur inf. a 0.6215
```

Exemple de fonction en statistique

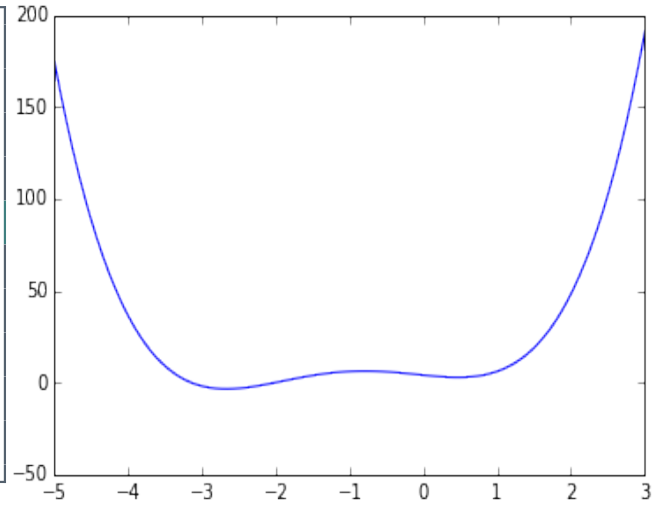
Test d'adéquation à la loi normale (ou autre):
Objectif : vérifier que la distribution est compatible avec la loi Normale

```
# Test de normalite d'Agostino
ag = stat.normaltest(d) # message d'avertissement, n est trop
    faible pour un test fiable
print(ag)
NormaltestResult(statistic=0.714391, pvalue=0.699635) #statistique
    de test et p-value (si p-value < alpha, rejet de l'hyp. de
    normalite)

# Test de Normalite Shapiro-Wilks
sp = stat.shapiro(d)
print(sp)
(0.9613385200500488, 0.627672016620636)   # statistique et p-value

#Test d'adequation d'Anderson-Darling
ad = stat.anderson(d,dist="norm") # test possible pour autre loi
    que " norm "
print(ad)
AndersonResult(statistic=0.340296323686203 55,
    critical_values=array([ 0.503,  0.573,  0.687,  0.802,
    0.954]), significance_level=array([ 15. ,  10. ,   5. ,   2.5,
      1. ]))
# stat de test, seuils critiques pour chaque niveau de risque, on
    constate ici que la p-value est sup. a 15%
```

Optimisation

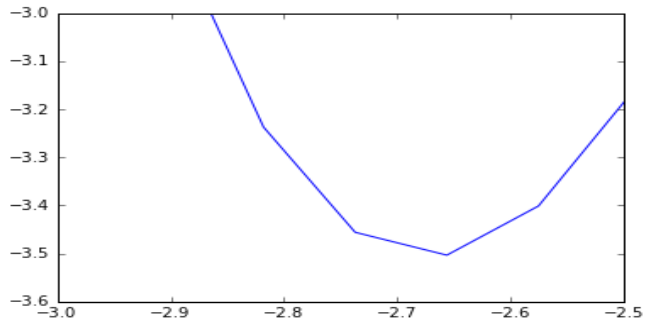
Objectif : trouver les minima ou maxima d'une fonction

```
# On import tout d'abord le module
from scipy import optimize

# On cherche le minimum de la fonction
    suivante :
def f(x):
    return 4*x**3 + (x-2)**2 + x**4

x = np.linspace(-5, 3, 100)
plt.plot(x, f(x))
```




```
x_min = optimize.fmin_bfgs(f, x0=-3)
x_min
```

```
Optimization terminated successfully.
         Current function value: -3.506641
         Iterations: 5
         Function evaluations: 24
         Gradient evaluations: 8
Out[41]:
array([-2.67298149])
```



À vous de jouer