

# SciPy - Librairie d'algorithmes pour le calcul scientifique en Python

Alexandre Gramfort : alexandre.gramfort@telecom-paristech.fr

Slim Essid : slim.essid@telecom-paristech.fr

adapté du travail de J.R. Johansson (robert@riken.jp) <http://dml.riken.jp/~rob/> (<http://dml.riken.jp/~rob/>)

## Introduction

SciPy s'appuie sur NumPy.

SciPy fournit des implémentations efficaces d'algorithmes standards.

Certains des sujets couverts par SciPy:

- Fonctions Spéciales ([scipy.special](http://docs.scipy.org/doc/scipy/reference/special.html) (<http://docs.scipy.org/doc/scipy/reference/special.html>))
- Intégration ([scipy.integrate](http://docs.scipy.org/doc/scipy/reference/integrate.html) (<http://docs.scipy.org/doc/scipy/reference/integrate.html>))
- Optimisation ([scipy.optimize](http://docs.scipy.org/doc/scipy/reference/optimize.html) (<http://docs.scipy.org/doc/scipy/reference/optimize.html>))
- Interpolation ([scipy.interpolate](http://docs.scipy.org/doc/scipy/reference/interpolate.html) (<http://docs.scipy.org/doc/scipy/reference/interpolate.html>))
- Transformées de Fourier ([scipy.fftpack](http://docs.scipy.org/doc/scipy/reference/fftpack.html) (<http://docs.scipy.org/doc/scipy/reference/fftpack.html>))
- Traitement du Signal ([scipy.signal](http://docs.scipy.org/doc/scipy/reference/signal.html) (<http://docs.scipy.org/doc/scipy/reference/signal.html>))
- Algèbre Linéaire ([scipy.linalg](http://docs.scipy.org/doc/scipy/reference/linalg.html) (<http://docs.scipy.org/doc/scipy/reference/linalg.html>))
- Matrices *Sparse* et Algèbre Linéaire Sparse ([scipy.sparse](http://docs.scipy.org/doc/scipy/reference/sparse.html) (<http://docs.scipy.org/doc/scipy/reference/sparse.html>))
- Statistiques ([scipy.stats](http://docs.scipy.org/doc/scipy/reference/stats.html) (<http://docs.scipy.org/doc/scipy/reference/stats.html>))
- Traitement d'images N-dimensionnelles ([scipy.ndimage](http://docs.scipy.org/doc/scipy/reference/ndimage.html) (<http://docs.scipy.org/doc/scipy/reference/ndimage.html>))
- Lecture/Ecriture Fichiers IO ([scipy.io](http://docs.scipy.org/doc/scipy/reference/io.html) (<http://docs.scipy.org/doc/scipy/reference/io.html>))

Durant ce cours on abordera certains de ces modules.

Pour utiliser un module de SciPy dans un programme Python il faut commencer par l'importer.

Voici un exemple avec le module *linalg*

```
In [1]: from scipy import linalg
```

On aura besoin de NumPy:

```
In [2]: import numpy as np
```

Et de matplotlib/pylab:

```
In [3]: # et JUSTE POUR MOI (pour avoir les figures dans le notebook)
%matplotlib inline
import matplotlib.pyplot as plt
```

## Fonctions Spéciales

Un grand nombre de fonctions importantes, notamment en physique, sont disponibles dans le module *scipy.special*

Pour plus de détails: <http://docs.scipy.org/doc/scipy/reference/special.html#module-scipy.special>  
(<http://docs.scipy.org/doc/scipy/reference/special.html#module-scipy.special>).

Un exemple avec les fonctions de Bessel:

```
In [4]: # jn : Bessel de premier type
# yn : Bessel de deuxième type
from scipy.special import jn, yn
```

```
In [5]: jn?
```

```
In [6]: n = 0      # ordre
x = 0.0

# Bessel de premier type
print("J_%d(%s) = %f" % (n, x, jn(n, x)))

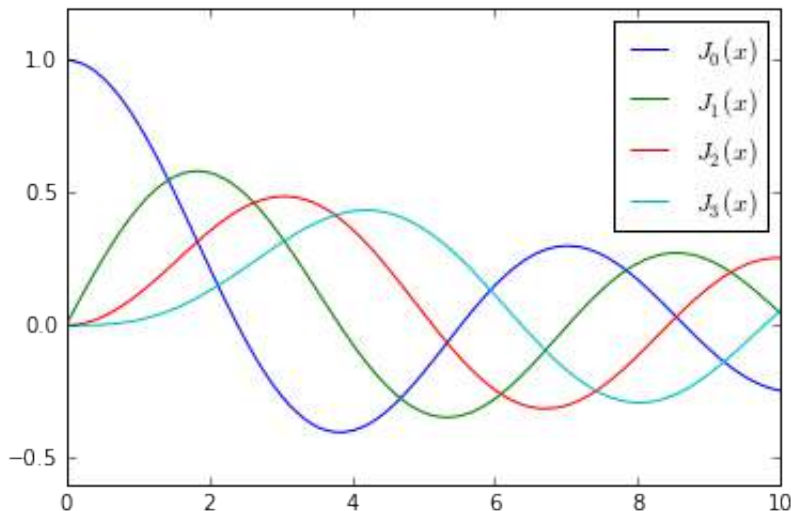
x = 1.0
# Bessel de deuxième type
print("Y_%d(%s) = %f" % (n, x, yn(n, x)))

J_0(0.0) = 1.000000
Y_0(1.0) = 0.088257
```

```
In [7]: x = np.linspace(0, 10, 100)

for n in range(4):
    plt.plot(x, jn(n, x), label=r"$J_{%d}(x)$" % n)
plt.legend()
```

Out[7]: <matplotlib.legend.Legend at 0x10f0db550>



```
In [8]: from scipy import special
special?
```

## Intégration

### intégration numérique

L'évaluation numérique de:

$$\int_a^b f(x)dx$$

est nommée *quadrature* (abbr. quad). SciPy fournit différentes fonctions: par exemple quad, dblquad et tplquad pour les intégrales simples, doubles ou triples.

```
In [9]: from scipy.integrate import quad, dblquad, tplquad
```

```
In [10]: quad?
```

L'usage de base:

```
In [11]: # soit une fonction f
def f(x):
    return x
```

```
In [13]: a, b = 1, 2 # intégrale entre a et b

val, abserr = quad(f, a, b)

print("intégrale =", val, ", erreur =", abserr)

('intégrale =', 1.5, ', erreur =', 1.6653345369377348e-14)
```

## EXERCICE: Intégrer la fonction de Bessel $J_n$ d'ordre 3 entre 0 et 10

In [ ]:

Exemple intégrale double:

$$\int_{x=1}^2 \int_{y=1}^x (x + y^2) dx dy$$

In [14]: ddblquad?

```
In [16]: def f(y, x):
    return x + y**2

def gfun(x):
    return 1

def hfun(x):
    return x

print(ddblquad(f, 1, 2, gfun, hfun))

(1.7500000000000002, 1.9428902930940243e-14)
```

## Equations différentielles ordinaires (EDO)

SciPy fournit deux façons de résoudre les EDO: Une API basée sur la fonction `odeint`, et une API orientée-objet basée sur la classe `ode`.

`odeint` est plus simple pour commencer.

Commençons par l'importer:

```
In [17]: from scipy.integrate import odeint
```

Un système d'EDO se formule de la façon standard:

$$y' = f(y, t)$$

avec

$$y = [y_1(t), y_2(t), \dots, y_n(t)]$$

et  $f$  est une fonction qui fournit les dérivées des fonctions  $y_i(t)$ . Pour résoudre une EDO il faut spécifier  $f$  et les conditions initiales,  $y(0)$ .

Une fois définies, on peut utiliser `odeint`:

$$y\_t = \text{odeint}(f, y\_0, t)$$

où  $t$  est un NumPy *array* des coordonnées en temps où résoudre l'EDO.  $y\_t$  est un array avec une ligne pour chaque point du temps  $t$ , et chaque colonne correspond à la solution  $y\_i(t)$  à chaque point du temps.

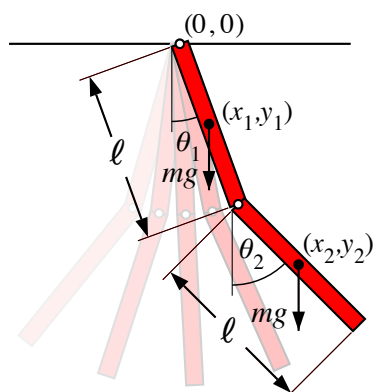
### Exemple: double pendule

Description: [http://en.wikipedia.org/wiki/Double\\_pendulum](http://en.wikipedia.org/wiki/Double_pendulum)

([http://en.wikipedia.org/wiki/Double\\_pendulum](http://en.wikipedia.org/wiki/Double_pendulum))

```
In [18]: from IPython.core.display import Image
Image(url='http://upload.wikimedia.org/wikipedia/commons/c/c9/Double-compound-pendulum-dimensioned.svg')
```

Out[18]:



Les équations du mouvement du pendule sont données sur la page wikipedia:

$$\dot{\theta}_1 = \frac{6}{m\ell^2} \frac{2p_{\theta_1} - 3 \cos(\theta_1 - \theta_2)p_{\theta_2}}{16 - 9 \cos^2(\theta_1 - \theta_2)}$$

$$\dot{\theta}_2 = \frac{6}{m\ell^2} \frac{8p_{\theta_2} - 3 \cos(\theta_1 - \theta_2)p_{\theta_1}}{16 - 9 \cos^2(\theta_1 - \theta_2)}$$

$$\dot{p}_{\theta_1} = -\frac{1}{2}m\ell^2 \left[ \dot{\theta}_1 \dot{\theta}_2 \sin(\theta_1 - \theta_2) + 3 \frac{g}{\ell} \sin \theta_1 \right]$$

$$\dot{p}_{\theta_2} = -\frac{1}{2}m\ell^2 \left[ -\dot{\theta}_1 \dot{\theta}_2 \sin(\theta_1 - \theta_2) + \frac{g}{\ell} \sin \theta_2 \right]$$

où les  $p_{\theta_i}$  sont les moments d'inertie. Pour simplifier le code Python, on peut introduire la variable  $x = [\theta_1, \theta_2, p_{\theta_1}, p_{\theta_2}]$

$$\dot{x}_1 = \frac{6}{m\ell^2} \frac{2x_3 - 3 \cos(x_1 - x_2)x_4}{16 - 9 \cos^2(x_1 - x_2)}$$

$$\dot{x}_2 = \frac{6}{m\ell^2} \frac{8x_4 - 3 \cos(x_1 - x_2)x_3}{16 - 9 \cos^2(x_1 - x_2)}$$

$$\dot{x}_3 = -\frac{1}{2}m\ell^2 \left[ \dot{x}_1 \dot{x}_2 \sin(x_1 - x_2) + 3 \frac{g}{\ell} \sin x_1 \right]$$

$$\dot{x}_4 = -\frac{1}{2}m\ell^2 \left[ -\dot{x}_1 \dot{x}_2 \sin(x_1 - x_2) + \frac{g}{\ell} \sin x_2 \right]$$

```
In [19]: g = 9.82
L = 0.5
m = 0.1

def dx(x, t):
    """The right-hand side of the pendulum ODE"""
    x1, x2, x3, x4 = x[0], x[1], x[2], x[3]

    dx1 = 6.0/(m*L**2) * (2 * x3 - 3 * np.cos(x1-x2) * x4)/(16 - 9
    * np.cos(x1-x2)**2)
    dx2 = 6.0/(m*L**2) * (8 * x4 - 3 * np.cos(x1-x2) * x3)/(16 - 9
    * np.cos(x1-x2)**2)
    dx3 = -0.5 * m * L**2 * ( dx1 * dx2 * np.sin(x1-x2) + 3 * (g/L)
    * np.sin(x1))
    dx4 = -0.5 * m * L**2 * (-dx1 * dx2 * np.sin(x1-x2) + (g/L) * n
    p.sin(x2))

    return [dx1, dx2, dx3, dx4]
```

```
In [20]: # on choisit une condition initiale
x0 = [np.pi/4, np.pi/2, 0, 0]
```

```
In [21]: # les instants du temps: de 0 à 10 secondes
t = np.linspace(0, 10, 250)
```

```
In [22]: # On résout
x = odeint(dx, x0, t)
print x.shape
```

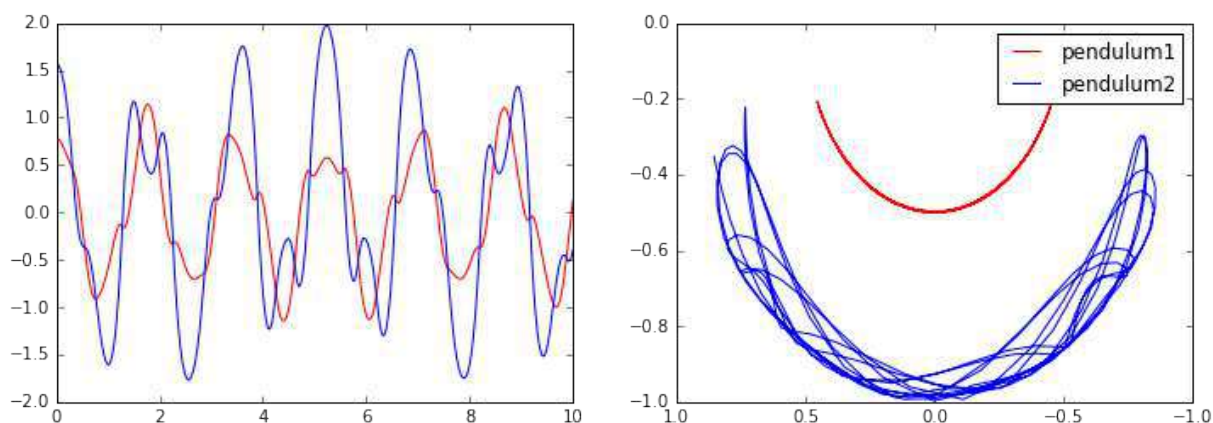
```
(250, 4)
```

```
In [23]: # affichage des angles en fonction du temps
fig, axes = plt.subplots(1,2, figsize=(12,4))
axes[0].plot(t, x[:, 0], 'r', label="theta1")
axes[0].plot(t, x[:, 1], 'b', label="theta2")

x1 = + L * np.sin(x[:, 0])
y1 = - L * np.cos(x[:, 0])
x2 = x1 + L * np.sin(x[:, 1])
y2 = y1 - L * np.cos(x[:, 1])

axes[1].plot(x1, y1, 'r', label="pendulum1")
axes[1].plot(x2, y2, 'b', label="pendulum2")
axes[1].set_ylim([-1, 0])
axes[1].set_xlim([1, -1])
plt.legend()
```

```
Out[23]: <matplotlib.legend.Legend at 0x112220f10>
```



## Transformées de Fourier

SciPy utilise la librairie [FFTPACK](http://www.netlib.org/fftpack/) (<http://www.netlib.org/fftpack/>) écrite en FORTRAN.

Commençons par l'import:

```
In [24]: from scipy import fftpack
```

Nous allons calculer les transformées de Fourier discrètes de fonctions spéciales:

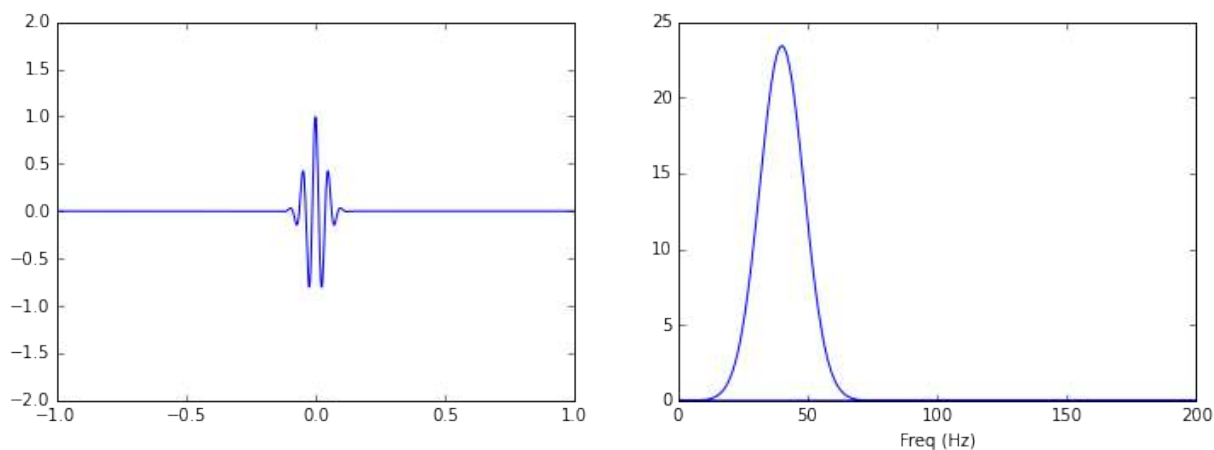
```
In [26]: from scipy.signal import gausspulse

t = np.linspace(-1, 1, 1000)
x = gausspulse(t, fc=20, bw=0.5)

# Calcul de la TFD
F = fftpack.fft(x)

# calcul des fréquences en Hz si on suppose un échantillonnage à 100
0Hz
freqs = fftpack.fftfreq(len(x), 1. / 1000.)
fig, axes = plt.subplots(1, 2, figsize=(12,4))
axes[0].plot(t, x) # plot du signal
axes[0].set_ylim([-2, 2])

axes[1].plot(freqs, np.abs(F)) # plot du module de la TFD
axes[1].set_xlim([0, 200])
# mask = (freqs > 0) & (freqs < 200)
# axes[0].plot(freqs[mask], abs(F[mask])) # plot du module de la TF
D
axes[1].set_xlabel('Freq (Hz)')
plt.show()
```



**EXERCICE : Le signal est réel du coup la TFD est symétrique. Afficher la TFD restreinte aux fréquences positives et la TFD restreinte aux fréquences entre 0 et 200Hz.**



## Algèbre linéaire

Le module de SciPy pour l'algèbre linéaire est `linalg`. Il inclut des routines pour la résolution des systèmes linéaires, recherche de vecteur/valeurs propres, SVD, Pivot de Gauss (LU, cholesky), calcul de déterminant etc.

Documentation : <http://docs.scipy.org/doc/scipy/reference/linalg.html>  
(<http://docs.scipy.org/doc/scipy/reference/linalg.html>)

### Résolution d'equations linéaires

Trouver  $x$  tel que:

$$Ax = b$$

avec  $A$  une matrice et  $x, b$  des vecteurs.

```
In [28]: A = np.array([[1,0,3], [4,5,12], [7,8,9]], dtype=np.float)
b = np.array([[1,2,3]], dtype=np.float).T
print(A)
print(b)
```

```
[[ 1.  0.  3.]
 [ 4.  5. 12.]
 [ 7.  8.  9.]]
[[ 1.]
 [ 2.]
 [ 3.]]
```

```
In [29]: from scipy import linalg
x = linalg.solve(A, b)
print(x)
```

```
[[ 0.8      ]
 [-0.4      ]
 [ 0.06666667]]
```

```
In [30]: print(x.shape)
print(b.shape)
```

```
(3, 1)
(3, 1)
```

```
In [31]: # Vérifier le résultat
```

### Valeurs propres et vecteurs propres

$$Av_n = \lambda_n v_n$$

avec  $v_n$  le  $n$ ème vecteur propre et  $\lambda_n$  la  $n$ ème valeur propre.

Les fonctions sont: eigvals et eig

```
In [32]: A = np.random.randn(3, 3)
```

```
In [33]: evals, evecs = linalg.eig(A)
```

```
In [34]: evals
```

```
Out[34]: array([-0.62446574+1.97855695j, -0.62446574-1.97855695j,  1.02893946
+0.j          ])
```

```
In [35]: evecs
```

```
Out[35]: array([[ -0.07263496+0.54831147j, -0.07263496-0.54831147j,  0.3783169
6+0.j          ],
 [ -0.80888273+0.j          , -0.80888273-0.j          , -0.4197660
1+0.j          ],
 [ 0.19786628+0.02522618j,  0.19786628-0.02522618j,  0.8250289
5+0.j          ]])
```

## EXERCICE : vérifier qu'on a bien des valeurs et vecteurs propres.

```
In [ ]:
```

Si A est symétrique

```
In [36]: A = A + A.T
# A += A.T # ATTENTION MARCHE PAS !!!!
evals = linalg.eigvalsh(A)
print(evals)
```

```
[-3.83544041  0.22487425  3.17058211]
```

```
In [37]: print(linalg.eigh(A))
```

```
(array([-3.83544041,  0.22487425,  3.17058211]), array([[ 0.80748919
, -0.43025042,  0.40354154],
 [ 0.40167574,  0.90204961,  0.15799714],
 [-0.43199283,  0.03451186,  0.90121647]]))
```

## Opérations matricielles

```
In [38]: # inversion
         linalg.inv(A)
```

```
Out[38]: array([[ 0.70455316, -1.79034249,  0.13962165],
                [-1.79034249,  3.58424481,  0.22859031],
                [ 0.13962165,  0.22859031,  0.21280511]])
```

```
In [40]: # vérifier
```

```
In [41]: # déterminant
         linalg.det(A)
```

```
Out[41]: -2.734600998424457
```

```
In [42]: # normes
         print(linalg.norm(A, ord='fro')) # frobenius
         print(linalg.norm(A, ord=2))
         print(linalg.norm(A, ord=np.inf))
```

```
4.98134143463
3.83544041032
5.55970464297
```

## EXERCICE : Vérifier les résultats

La norme infinie est la norme infinie de la norme 1 de chaque ligne.

```
In [ ]:
```

## Optimisation

**Objectif:** trouver les minima ou maxima d'une fonction

Doc : [http://scipy-lectures.github.com/advanced/mathematical\\_optimization/index.html](http://scipy-lectures.github.com/advanced/mathematical_optimization/index.html) ([http://scipy-lectures.github.com/advanced/mathematical\\_optimization/index.html](http://scipy-lectures.github.com/advanced/mathematical_optimization/index.html))

On commence par l'import

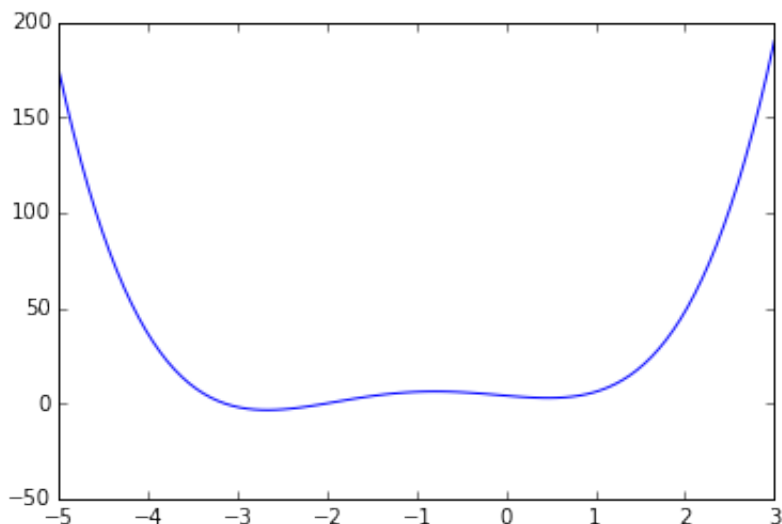
```
In [43]: from scipy import optimize
```

## Trouver un minimum

```
In [44]: def f(x):  
         return 4*x**3 + (x-2)**2 + x**4
```

```
In [45]: x = np.linspace(-5, 3, 100)  
         plt.plot(x, f(x))
```

```
Out[45]: [<matplotlib.lines.Line2D at 0x1139e10d0>]
```



Nous allons utiliser la fonction `fmin_bfgs`:

```
In [46]: 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[46]: array([-2.67298149])
```

## Trouver les zéros d'une fonction

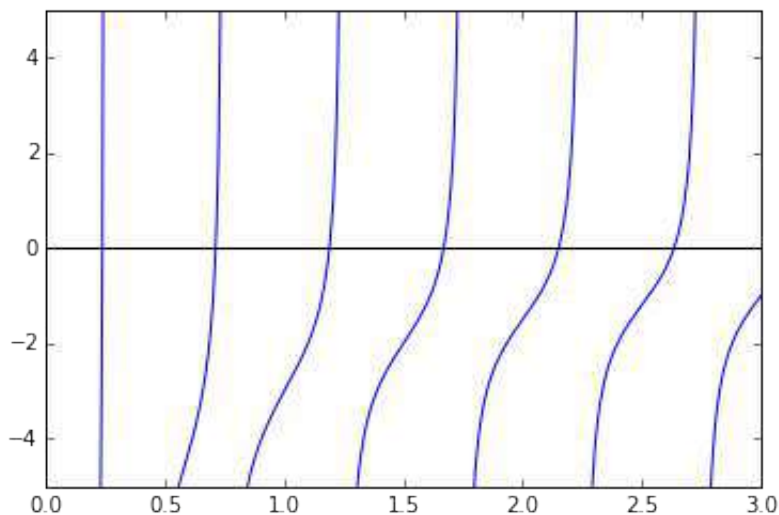
Trouver  $x$  tel que  $f(x) = 0$ . On va utiliser `fsolve`.

```
In [47]: omega_c = 3.0  
         def f(omega):  
             return np.tan(2*np.pi*omega) - omega_c/omega
```

```
In [48]: x = np.linspace(0, 3, 1000)
y = f(x)
mask = np.where(abs(y) > 50)
x[mask] = y[mask] = np.nan # get rid of vertical line when the func
tion flip sign
plt.plot(x, y)
plt.plot([0, 3], [0, 0], 'k')
plt.ylim(-5,5)
```

-c:3: RuntimeWarning: divide by zero encountered in divide

Out[48]: (-5, 5)



```
In [49]: np.unique(
    (optimize.fsolve(f, np.linspace(0.2, 3, 40))*1000).astype(int)
) / 1000.
```

Out[49]: array([ 0.237, 0.712, 1.189, 1.669, 2.15 , 2.635, 3.121, 3.61  
])

```
In [50]: optimize.fsolve(f, 0.72)
```

Out[50]: array([ 0.71286972])

```
In [51]: optimize.fsolve(f, 1.1)
```

Out[51]: array([ 1.18990285])

## Estimation de paramètres de fonctions

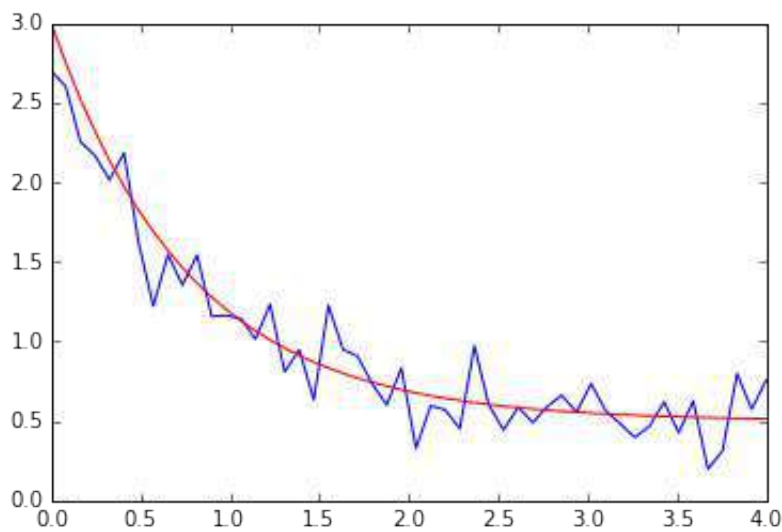
```
In [52]: from scipy.optimize import curve_fit

def f(x, a, b, c):
    """
     $f(x) = a \exp(-bx) + c$ 
    """
    return a*np.exp(-b*x) + c

x = np.linspace(0, 4, 50)
y = f(x, 2.5, 1.3, 0.5)
yn = y + 0.2*np.random.randn(len(x)) # ajout de bruit
```

```
In [53]: plt.plot(x, yn)
plt.plot(x, y, 'r')
```

```
Out[53]: [<matplotlib.lines.Line2D at 0x1139edb90>]
```



```
In [54]: (a, b, c), _ = curve_fit(f, x, yn)
print(a, b, c)

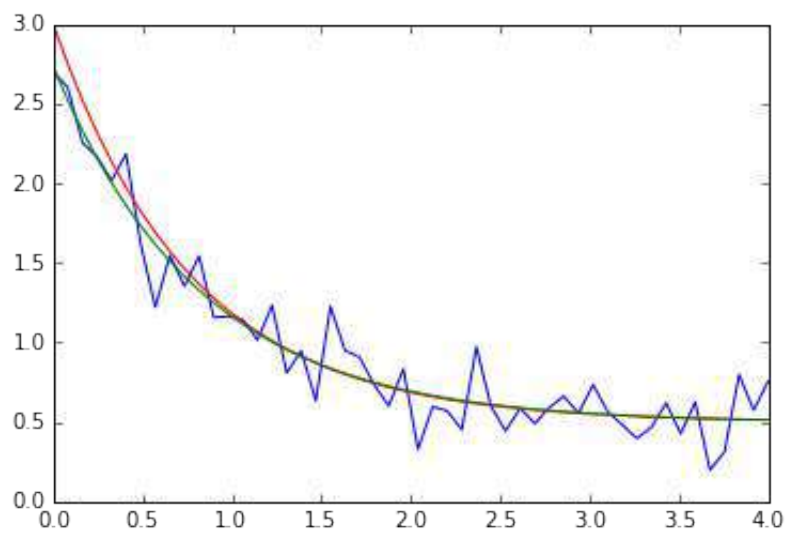
2.23756960904 1.21041192888 0.49442120282
```

```
In [55]: curve_fit?
```

On affiche la fonction estimée:

```
In [56]: plt.plot(x, yn)
plt.plot(x, y, 'r')
plt.plot(x, f(x, a, b, c))
```

```
Out[56]: [<matplotlib.lines.Line2D at 0x113d25690>]
```

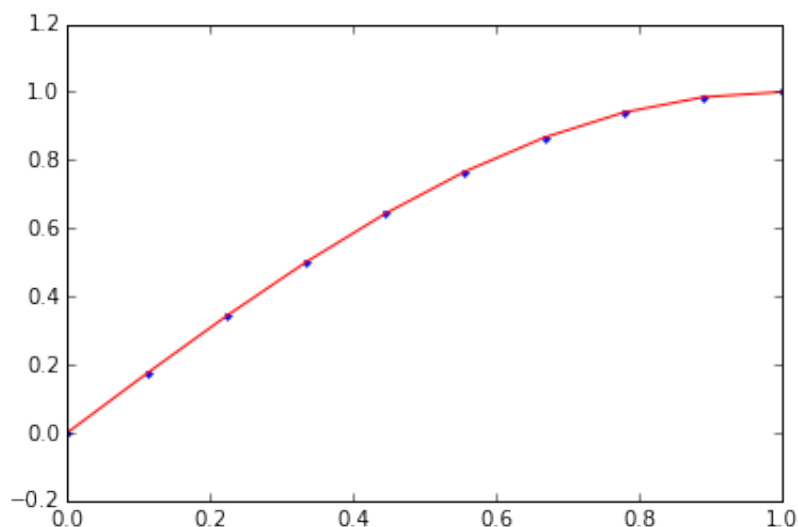


Dans le cas de polynôme on peut le faire directement avec NumPy

```
In [57]: x = np.linspace(0,1,10)
y = np.sin(x * np.pi / 2.)
line = np.polyfit(x, y, deg=10)
plt.plot(x, y, '.')
plt.plot(x, np.polyval(line,x), 'r')
# xx = np.linspace(-5,4,100)
# plt.plot(xx, np.polyval(line,xx), 'g')
```

```
/Users/alex/anaconda/lib/python2.7/site-packages/numpy/lib/polynomia
l.py:594: RankWarning: Polyfit may be poorly conditioned
warnings.warn(msg, RankWarning)
```

```
Out[57]: [<matplotlib.lines.Line2D at 0x113d8f550>]
```



## Interpolation

```
In [58]: from scipy.interpolate import interp1d
```

```
In [59]: def f(x):
return np.sin(x)
```

```
In [60]: n = np.arange(0, 10)
x = np.linspace(0, 9, 100)

y_meas = f(n) + 0.1 * np.random.randn(len(n)) # ajout de bruit
y_real = f(x)

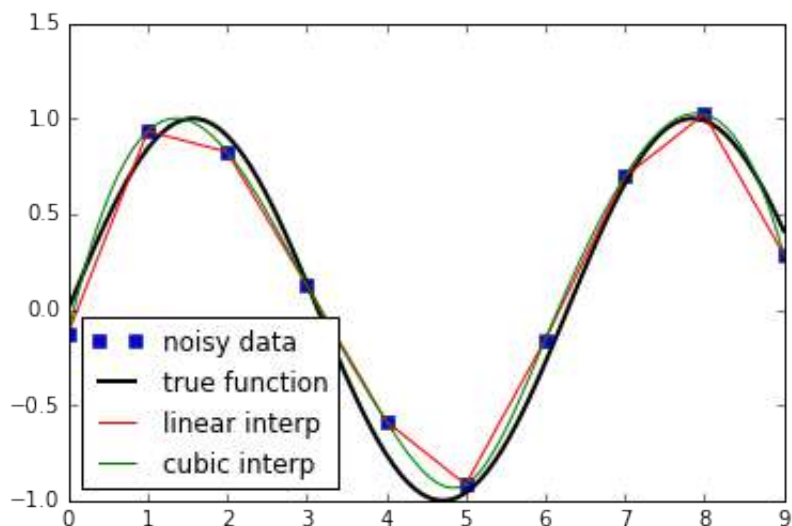
linear_interpolation = interp1d(n, y_meas)
y_interp1 = linear_interpolation(x)

cubic_interpolation = interp1d(n, y_meas, kind='cubic')
y_interp2 = cubic_interpolation(x)
```



```
In [61]: from scipy.interpolate import barycentric_interpolate, BarycentricI
nterpolator
BarycentricInterpolator??
```

```
In [62]: plt.plot(x, y_meas, 'bs', label='noisy data')
plt.plot(x, y_real, 'k', lw=2, label='true function')
plt.plot(x, y_interp1, 'r', label='linear interp')
plt.plot(x, y_interp2, 'g', label='cubic interp')
plt.legend(loc=3);
```



## Images

```
In [63]: from scipy import ndimage
from scipy import misc
img = misc.lena()
print img
type(img), img.dtype, img.ndim, img.shape
```

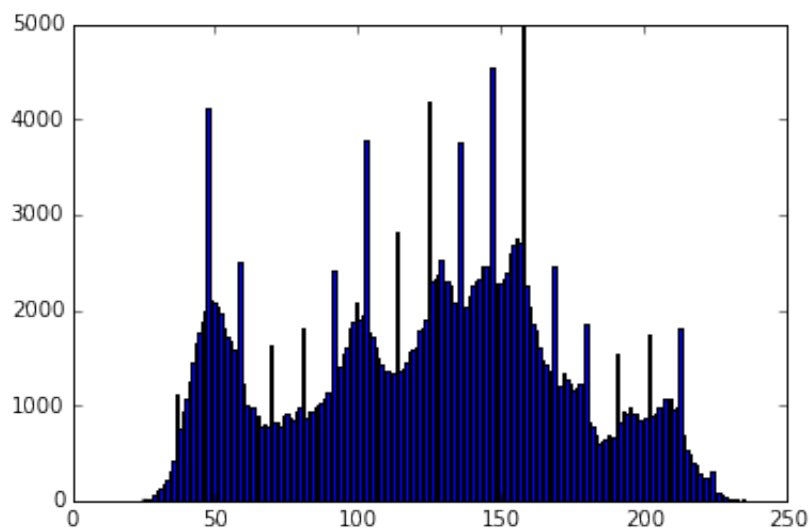
```
[[162 162 162 ..., 170 155 128]
 [162 162 162 ..., 170 155 128]
 [162 162 162 ..., 170 155 128]
 ...,
 [ 43  43  50 ..., 104 100  98]
 [ 44  44  55 ..., 104 105 108]
 [ 44  44  55 ..., 104 105 108]]
```

```
Out[63]: (numpy.ndarray, dtype('int64'), 2, (512, 512))
```

```
In [64]: plt.imshow(img, cmap=plt.cm.gray)
plt.axis('off')
plt.show()
```



```
In [65]: _ = plt.hist(img.reshape(img.size), 200)
```



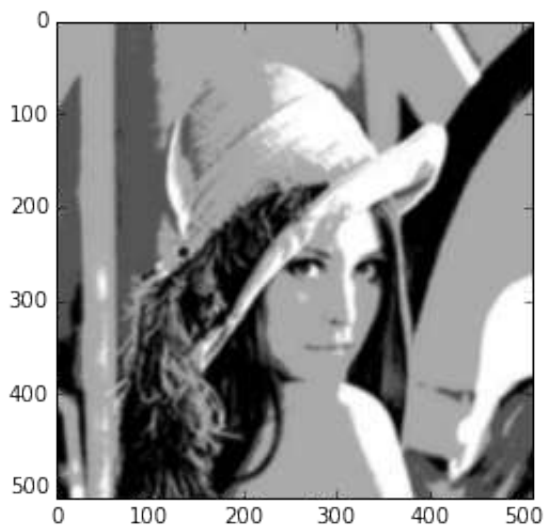
```
In [66]: img[img < 70] = 50
img[(img >= 70) & (img < 110)] = 100
img[(img >= 110) & (img < 180)] = 150
img[(img >= 180)] = 200
plt.imshow(img, cmap=plt.cm.gray)
plt.axis('off')
plt.show()
```



### Ajout d'un flou

```
In [67]: img_flou = ndimage.gaussian_filter(img, sigma=2)
plt.imshow(img_flou, cmap=plt.cm.gray)
```

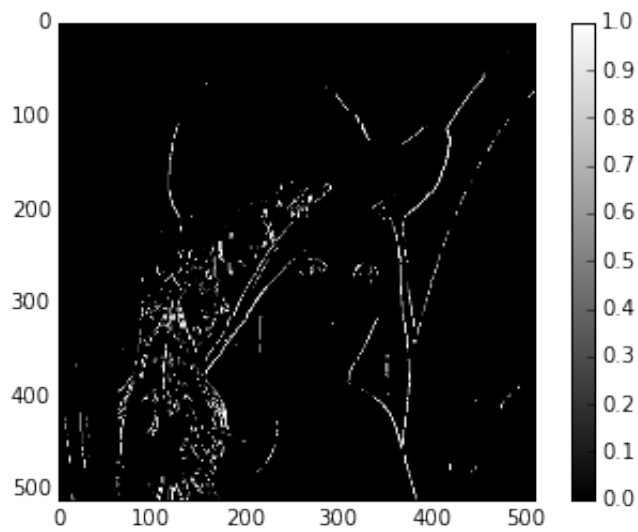
Out[67]: <matplotlib.image.AxesImage at 0x116e45190>



### Application d'un filtre

```
In [69]: img_sobel = ndimage.filters.sobel(img)
plt.imshow(np.abs(img_sobel) > 200, cmap=plt.cm.gray)
plt.colorbar()
```

Out[69]: <matplotlib.colorbar.Colorbar instance at 0x11793f680>

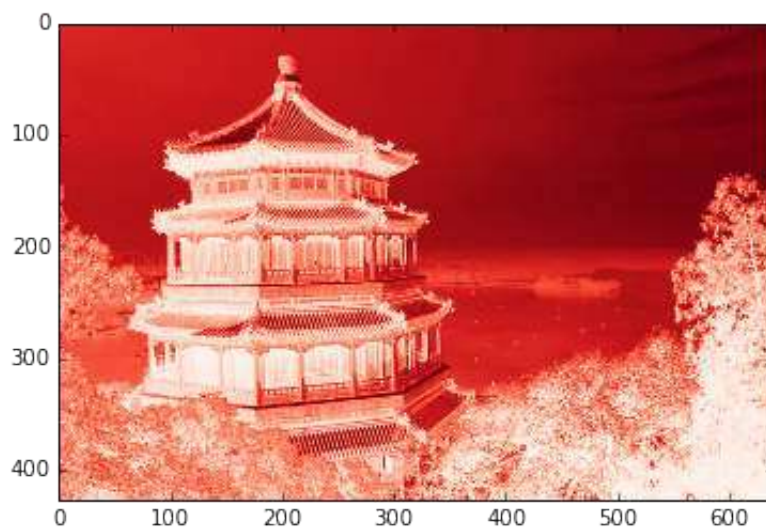


Accéder aux couches RGB d'une image:

```
In [70]: img = ndimage.imread('china.jpg')
print(img.shape)
plt.imshow(img[:, :, 0], cmap=plt.cm.Red)
```

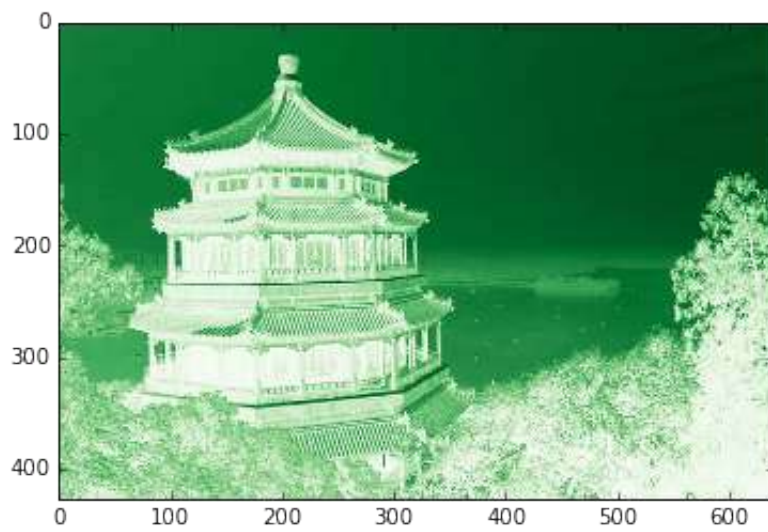
(427, 640, 3)

Out[70]: <matplotlib.image.AxesImage at 0x117dcb890>



```
In [71]: plt.imshow(img[:, :, 1], cmap=plt.cm.Greens)
```

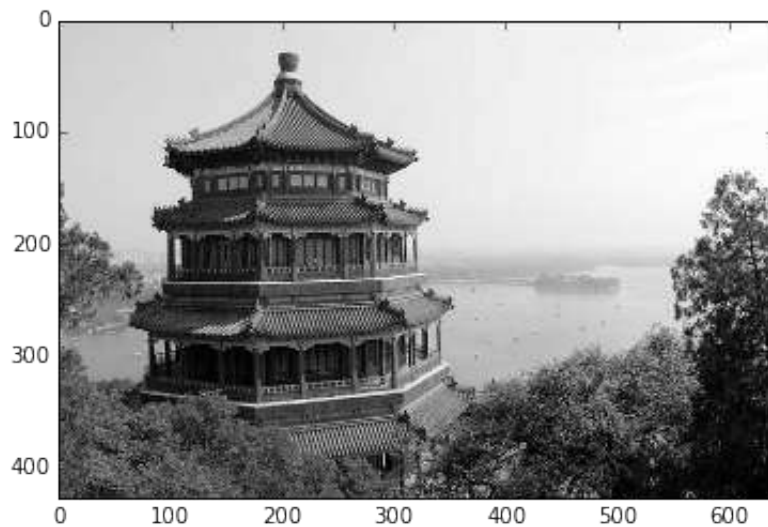
```
Out[71]: <matplotlib.image.AxesImage at 0x119159610>
```



Conversion de l'image en niveaux de gris et affichage:

```
In [72]: a = np.mean(img, axis=2)
plt.imshow(a, cmap=plt.cm.gray)
```

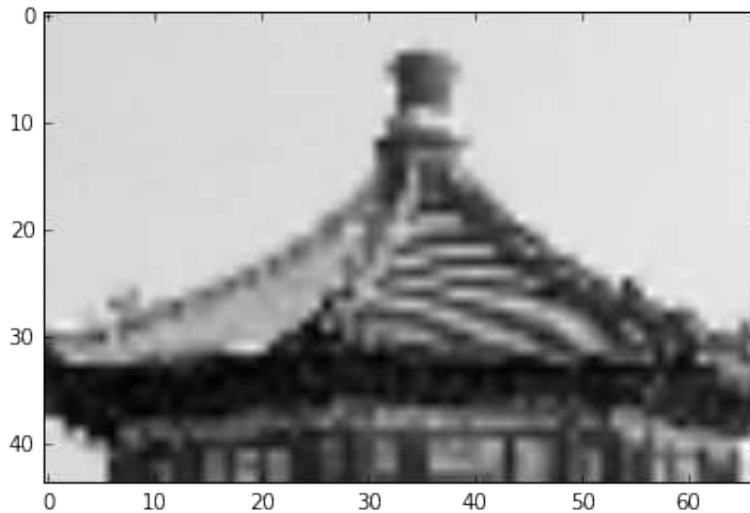
```
Out[72]: <matplotlib.image.AxesImage at 0x1134d71d0>
```



Observer le repliement spectral (aliasing)

```
In [73]: a = a[20:150, 100:300]
plt.imshow(a[:, :3, ::3], cmap=plt.cm.gray)
```

```
Out[73]: <matplotlib.image.AxesImage at 0x116b791d0>
```



## Pour aller plus loin

- <http://www.scipy.org> (<http://www.scipy.org>) - The official web page for the SciPy project.
- <http://docs.scipy.org/doc/scipy/reference/tutorial/index.html> (<http://docs.scipy.org/doc/scipy/reference/tutorial/index.html>) - A tutorial on how to get started using SciPy.
- <https://github.com/scipy/scipy/> (<https://github.com/scipy/scipy/>) - The SciPy source code.
- <http://scipy-lectures.github.io> (<http://scipy-lectures.github.io>)

```
In [ ]:
```