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

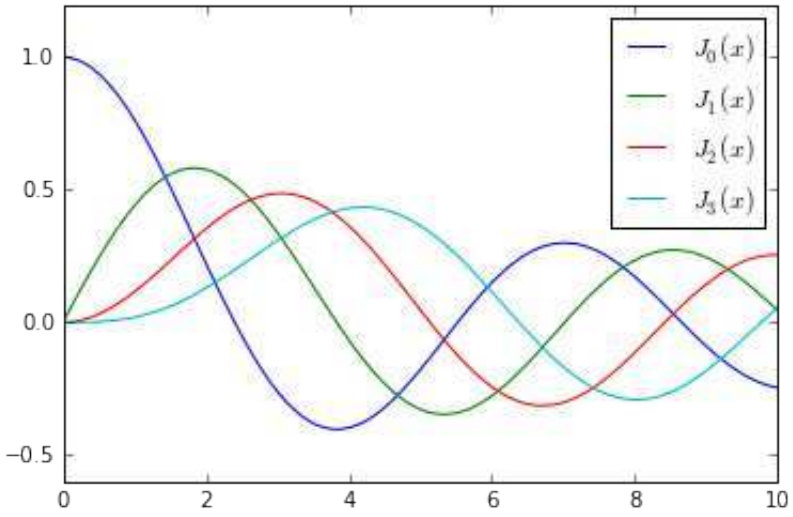
In [5]:
```
jn?
```

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 numerique

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)
```

```
('int\xc3\xa9grale =', 1.5, ', erreur =', 1.6653345369377348e-14)
```

EXERCICE: Intégrer la fonction de Bessel `jn` 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]:
```
dblquad?
```

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

def gfun(x):
    return 1

def hfun(x):
    return x

print(dblquad(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

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

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 = odeint(f, y_0, t)
```

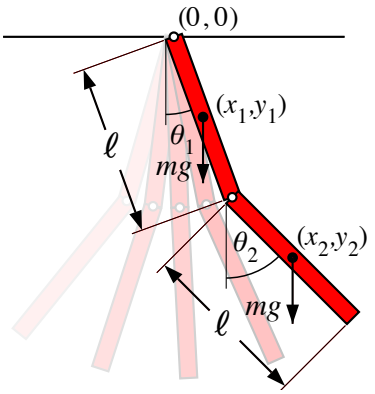
où `t` est un NumPy *array* des coordonnées en temps où résoudre l'EDO. `y_t` est un array avec une ligne pour chaque point du temps `t`, et chaque colonne correspond à la solution `y_i(t)` à chaque point du temps.

Exemple: double pendule

Description: http://en.wikipedia.org/wiki/Double_pendulum (http://en.wikipedia.org/wiki/Double_pendulum)

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:

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

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

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

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

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

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

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

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

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

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)
```

Ceci est un exemple, cliquez sur le lien de téléchargement pour obtenir le cours complet.