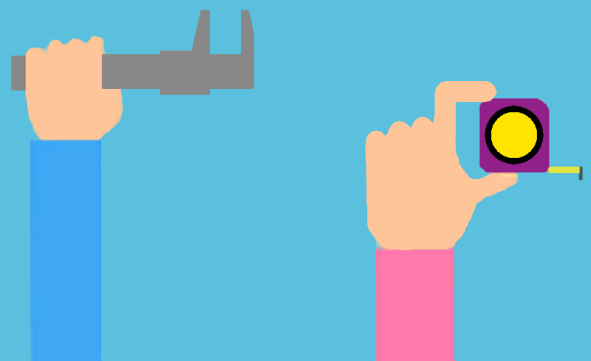
Fonctions complémentaires de Numpy

Renaud Costadoat
Lycée Dorian

DORIAN



Introduction

Les bibliothèques NumPy (http://www.numpy.org/) et SciPy (http://www.scipy.org/) permettent d'effectuer des calculs numériques avec Python.

Il faut au départ importer ces packages avec l'instruction suivante :

```
>>> import numpy as np
>>> import scipy.optimize as resol
>>> import scipy.integrate as integr
```

Résolution approchée d'équations

Pour résoudre une équation du type $f(x) = 0$ où f est une fonction d'une variable réelle, on peut utiliser la fonction `fsolve` . Lorsqu'il existe plusieurs solutions, il est possible de donner une estimation d'une solution afin de tendre vers ce résultat.

```
def f(x) :
    return x**2 - 3

>>> resol.fsolve(f, -1.)
[-1.73205081]
>>> resol.fsolve(f, 1.)
[ 1.73205081]
```

Résolution approchée d'équations

Dans le cas d'une fonction f à valeurs vectorielles, on utilise la fonction `root` . Par exemple, pour résoudre le système non linéaire: $\begin{cases} x^2 - y^2 = 2 \\ 2x - y - 1 = 2 \end{cases}$

```
def f(v):
   return v[0]**2-v[1]**2-2, 2*v[0]-v[1]-3
>>> sol=resol.root(f,[0,0])
>>> sol.success
True
>>> sol.x
[ 1.42264973 -0.15470054]
>>> sol=resol.root(f,[3,3])
>>> sol.success
True
>>> sol.x
[ 2.57735027  2.15470054]
```

Calcul approché d'intégrales

La fonction `quad` du module `scipy.integrate` permet de calculer la valeur approchée d'intégrales. Elle affiche l'erreur d'approximation.

```
def f(x) :
    return x**(-3)

>>> integr.quad(f, 1, 3)
(0.4444444444444444, 4.784195109973667e-11)
>>> integr.quad(f, 1, np.inf)
(0.5, 5.551115123125783e-15)
```

Calcul approché d'équations différentielles

Pour résoudre une équation différentielle du premier ordre ($x'(t) = f(x,t)$), on peut utiliser la fonction `odeint` du module `scipy.integrate`.

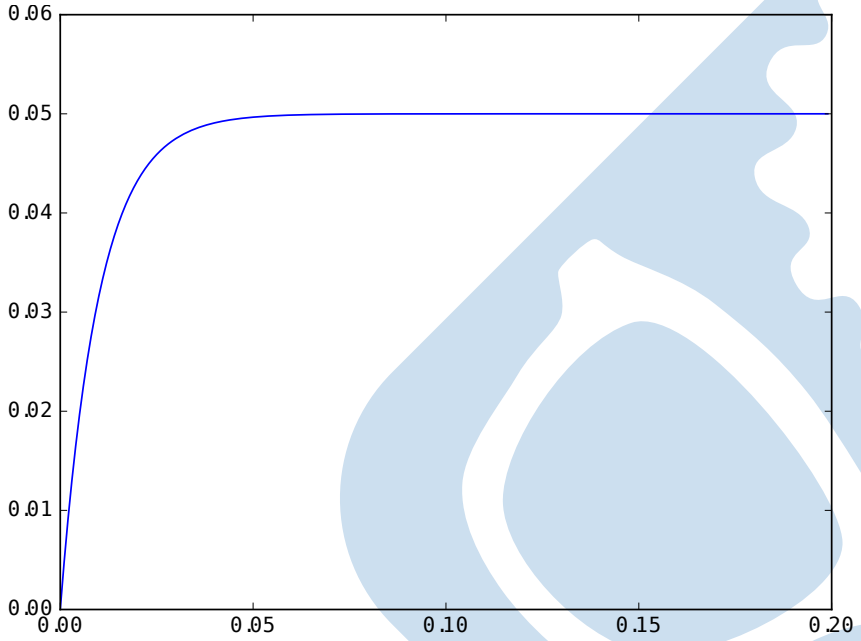
Cette fonction nécessite une liste de valeurs de t, commençant en t_0, et une condition initiale x_0. La fonction renvoie les valeurs approchées des solutions.

Par exemple, dans le but de déterminer la tension aux bornes d'un condensateur dans un circuit RC. Il est possible d'utiliser le code suivant.

```
E=5 #Tension en volt
R=200 #Résistance en ohm
C=50*10**(-6) #Capacité du condensateur en Coulomb
def f(u, t) :
   return E-u/(R*C)
```

Calcul approché d'équations différentielles

```
>>> T=np.arange(0,0.2,0.01)
>>> X= integr.odeint(f,0,T)
>>> X
array([[ 0.        ],
       [ 0.03160602],
       [ 0.04323323],
       [ 0.04751065],....
```



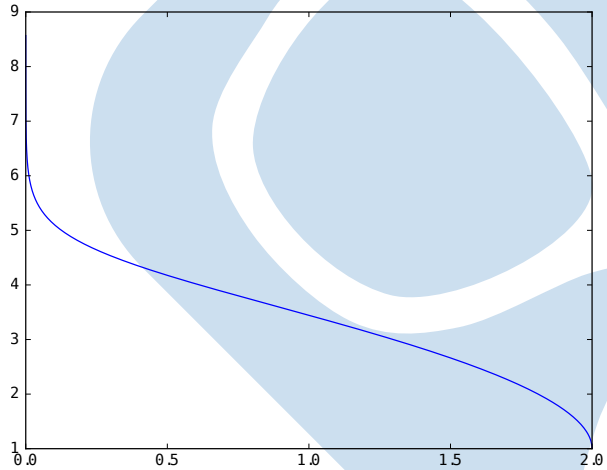
Question: Coder la résolution du mouvement oscillatoire d'une masse liée à un ensemble amortisseur/ressort.

Calcul approché d'équations différentielles

Il est alors possible de résoudre des systèmes d'équations différentielles.

$$\begin{cases} x'(t) = -t * x(t) \\ y'(t) = x(t) + 0.2 * y(t) \end{cases}$$, avec la condition initiale $x(0) = 2$, $y(0) = 1$.

```
def f(x, t) :
    f=np.array([-t*x[0], x[0]+0.2*x[1]])
    return f
T = np.arange(0, 5.01, 0.01)
X = integr.odeint(f, np.array([2.,1.]), T)
>>> X
[[  2.00000000e+00   1.00000000e+00]
 [  1.99990000e+00   1.02202166e+00]
 ...,
```



Le système d'ordre 1 satisfait par $X(t) = \begin{pmatrix} x(t) \\ x'(t) \end{pmatrix}$ permettra de résoudre une équation différentielle scalaire d'ordre 2 de solution x.

Nombres complexes

Avec Python le nombre imaginaire pur *i* se note `1j` . Les attributs `real` et `imag` permettent d'obtenir la partie réelle et la partie imaginaire. La fonction `abs` calcule son module.

```
>>> a = 6 + 2j
>>> b = 9 - 1j
>>> a*b
(56+12j)
>>> a.real
6.0
>>> a.imag
2.0
>>> abs(a)
6.324555320336759
```