Équations différentielles

déc. 19, 2018

Équations différentielles

La librairie `scipy.integrate` contient des fonctions pour résoudre les équations différentielles ordinaires, c'est à dire des équations de la forme :

$$\frac{dy}{dt} = f(t, y)$$

où $y(t)$ est une fonction de $\mathbb{R} \to \mathbb{R}^n$.

Regardons l'exemble $y' = -y$ avec $y(0) = 1$:

```
from scipy.integrate import ode

def f(t, y):
    return -y

r = ode(f).set_integrator('vode')
r.set_initial_value(y=1,t=0)

print(r.integrate(1))
print(r.t, r.y)
```

Quelques remarques :

— Plusieurs résolveurs peuvent être utilisés (voir le documentation). Chacun est en fait une librairie Fortran adaptée à scipy. Ils ont chacuns leurs propres parametères. Les plus communs sont la précision absolue (atol) et relative (rtol). Comparés aux résolveurs impératifs habituels (Euler ou Runge-Kutta), ceux de scipy sont capables d'adapter la taille de chaque étape afin d'obtenir la précision souhaitée.

— Pour des équations différencielles d'ordre élevé, l'astuce consiste à augmenter la dimension de y en rajoutant des fonctions intermédiaires qui sont les dérivées de la fonction initiales.

Par exemple, les solusions d'un oscillateur $y'' = -y$ avec pour conditions initiales $y(0) = 1$ et $y'(0) = 0$

```
from scipy.integrate import ode
from numpy import *

def f(t, Y):
    y, yprime=Y
    return array([yprime, -y])

r = ode(f).set_integrator('vode')
r.set_initial_value(array([1,0]),0)

print(r.integrate(pi))
```

Le pendule

On considère l'équation

$$\theta'' = -\sin\theta$$

Ecrire une fonction qui renvoie un tableau contenant la position et la vitesse angulaire du pendule pour N instants entre 0 et T.

La position initiale est $\theta = 0$, tracez les trajectoires dans l'espace des phases pour differentes valeurs de la vitesse angulaire initiale.

Nuage d'ions

On condidère N ions A_i de masse m_i et de charge q_i. On note $\vec{r_i}$ et $\vec{v_i}$ la position et la vitesse de l'ion A_i. Les ions dans dans un piège électrostatique, ils interagissent entre eux par la force de Coulomb.

La force électrostatique dérive d'un potentiel électrostatique :

$$V(\vec{r}) = \frac{1}{2}k_x r_x^2 + \frac{1}{2}k_y r_y^2 + \frac{1}{2}k_z r_z^2$$

La force de Coulomb s'écrira sous la forme :

$$\vec{f_i}(\vec{r_i}, \vec{r_j}) = \kappa \frac{q_i q_j}{|\vec{r_i} - \vec{r_j}|^3}(\vec{r_i} - \vec{r_j})$$

En plus de cette dynamique, certaine particules peuvent être refroidies par laser. On modélisera ce refroidissement par une force de friction unidirectionnelle, selon l'axe x :

$$F_{i,x} = -\alpha_i v_{i,x}$$

Pour simplifier et avoir une meilleure représentation graphique, on suprimera totalement la dimension z.

On utilisera des unités adimensionnées avec des constantes de l'ordre de 1. Par défaut on prendra : $m_i = 9$, $q_i = 1$, $k_x = 1$, $k_y = 1.3$

On part d'un nuage d'ion ayant une dispersion en posision et en vitesse donnée par une loi de Maxwell-Boltzmann (avec typiquement $k_B = 1$ et $T = 1$).

Après avoir écrit un programme permettant de résoudre les équations du mouvement, on pourra vérifier le code dans le cas sans amortissement :

— avec une particule, la trajectoire est eliptique dans le cas $k_x = k_y$.
— avec beaucoup de particules en interaction, l'énergie doit être conservée.

Cristal de Coulomb

On pourra introduire ensuite un amortissement (prendre typiquement $\alpha_i = 0.01$) et $N = 20$. Que ce passe-t-il au bout d'un certain temps ?

Enregistrer la position et vitesse des ions à M instants t_i. Après avoir fait le calcul, et en utilisant `matplotlib.animation.FuncAnimation` faire une animation (regarder sur internet comment faire !).

Refroidissement sympathique

On considère deux types d'ions : les premiers de masse 9 et charge 1 et les seconds de masse 1 et de charge 1. Seuls les premiers ions sont refroidis directement. Par interaction, les seconds ions seront refroidis eux aussi. C'est ce que l'on appelle le refroidissement sympathique. Visualiser sous forme d'une animation ce processus (on mettra deux couleurs).

Remarques

— Pour calculer la force dans l'équation différentielle, on pourra utiliser une boucle `for` et `numba.jit`.
— L'idéal sera de créer un objet qui fera le lien entre l'équation différentielle (bas niveau) et l'utilisateur. Cet objet réalisera la simulation et gardera les résultats. On pourra ensuite avoir des methodes permettant d'obtenir la temperature au cour du temps ou l'énergie au cour du temps. On pourra exporter sous forme d'une vidéo la simulation, etc.