Chapitre 10 : Simulation, résolution approchée d'équations différentielles avec `scipy`, `pylab`

1 Résolutions d'E.D. d'ordre 1

1.1 Point de terminologie et de forme

Terminologie : Une équation différentielle comme nous les connaissons est une *Equation Différentielle Ordinaire*, en anglais *Ordinary Differential Equation* donc `ode`.

Par opposition, pour les fonctions à plusieurs variables, les équations avec des dérivées partielles différentes seront appelées *Equations aux Dérivées partielles* (E.D.P. et en anglais `PDE`).

Forme : Une équation différentielle du premier ordre normalisée peut toujours s'écrire sous la forme :

$$y' = f(x,y) \text{ c'est-à-dire aussi } y'(x) = f(x,y(x))$$

Par exemple $y'(x) = 2x\sin(y(x)) + 1$ s'écrira $y' = f(x,y)$ avec $f(u,v) = 2u\sin(v) + 1$.

Parfois la fonction f ne dépend que de y, par exemple $y'(x) = 2.y(x)^2$. On dit que l'équation est *autonome* : ces équations ont des propriétés particulières, qui n'interviendront pas ici. Ici $f(u,v) = 2v^2$.

1.2 Point de départ de toutes les méthodes numériques

Toutes les méthodes numériques partent d'une C.I. $y(x_0) = y_0$.

On considère donc une E.D. $y'(x) = f(x,y(x))$ où $f : I \to \mathbb{R} \to \mathbb{R}$ et un couple $(x_0,y_0) \in I \times \mathbb{R}$. On cherche à construire une *solution approchée* du problème de Cauchy $\begin{cases} y'(x) = f(x,y(x)), \\ y(x_0) = y_0. \end{cases}$

1.3 Savoir faire indispensable : la méthode d'Euler

Avec les notations du paragraphe précédent, on va construire une fonction y *continue, affine par morceaux*, dont on espère qu'elle approche convenablement la « vraie » solution : on parle de *solution approchée.*

Pour chaque entier n, on va définir les valeurs y_i de y aux points $x_i = a + i\dfrac{b-a}{n}$, comme suit :

dans la formule $y'(x) = f(x,y(x))$ au point x_i, on va remplacer $y'(x_i)$ par le *taux de variation* $\dfrac{y(x_{i+1}) - y(x_i)}{x_{i+1} - x_i}$.

L'idée de la méthode d'Euler se résume alors à la formule suivante : $\boxed{\dfrac{y(x_{i+1}) - y(x_i)}{x_{i+1} - x_i} = f(x_i,y_i),}$

qui permet de calculer les valeurs $y_i = y(x_i)$ successivement puisque la formule précédente donne :

$$\boxed{y_{i+1} = y_i + f(x_i,y_i)(x_{i+1} - x_i).}$$

1.4 Exemple d'implémentation en Python (déjà vue en T.P.)

N.B. *On choisit, pour la commodité des applications à la physique, d'appeler plutôt t la variable des fonctions.*

Soit l'E.D. la plus simple possible ou presque $y'(t) = 2y(t)$ avec la C.I. $y(0) = 1$.
Disons qu'on travaille pour t dans $[0,5]$ avec un pas de temps de $p = 0,01s$.
On définit le vecteur `t` des temps successifs par :
`t=pl.arange(0,5.01,0.01)`

On définit alors le vecteur `y` dont les valeurs successives `y[i]` vérifient pour tout i jusqu'à l'avant dernier : `(y[i+1]-y[i])/p=2*y[i]`

Autrement dit, on a la relation de récurrence : `y[i+1]=(1+2p)*y[i]`
Avec le code (par exemple) :

```
for i in range(len(t)-1):
    y.append((1+2*p)*y[-1])# l'entrée précédente
```

Une fois qu'on a `t` et `y`, on peut tracer la courbe avec `pl.plot(t,y)`.

1.5 Des méthodes plus puissantes, avec `integrate.odeint`

Il existe des méthodes plus sophistiquées que la méthode d'Euler. Par exemple, dans le cas particulier des équations de la forme $y'(x) = f(x)$, la méthode d'Euler redonne la méthode des rectangles pour le calculs des intégrales. Or on connaît (ou connaîtra) des meilleures méthodes pour les calculs d'intégrales : nous avons vu la méthodes des trapèzes, et nous verrons des méthodes plus évoluées qui approchent la courbes par des courbes polynomiales de degré plus grand.. etc. De même pour les approximations de solutions d'E.D. il existe de bien meilleures méthodes.

De telles méthodes plus performantes sont implémentées dans `scipy` avec le sous-module `integrate`. On l'utilisera ici en important le module :

```
from scipy import integrate
```

Dans ce module, on utilisera la fonction `odeint`.

2 Comment utiliser `integrate.odeint` pour les E.D. du 1er ordre

On doit voir l'E.D. sous la forme $y'(t) = F(y(t), t)$.
Même si F ne dépend pas de t (équation *autonome*), on *doit* mettre la dépendance en `t` dans la définition de `F`.
L'ordre des arguments est `integrate.odeint(F,y0,t)`.

2.1 Reprise de l'exemple du 1.4

Voici le code pour résoudre la même équation qu'au § 1.4. avec `odeint`.

```
def F(y,t):
    # Noter qu'ici F ne dépend pas explicitement de t,
    # mais il est nécessaire de mentionner cette variable.
    return 2*y
t=pl.arange(0,5+p,p)
y0=1
y=integrate.odeint(F,y0,t)
pl.figure("Avec odeint")
pl.plot(t,y)
pl.show()
```

2.2 Un exemple non linéaire

Disons qu'on veuille résoudre l'E.D. $y'(t) = y(t)^2 \sin(t)$. Il se trouve qu'on sait résoudre encore formellement cette équation par séparations des variables. On considère la C.I. $y(0) = 0.3$. Le script suivant permet le tracé des courbes solutions sur $[0, 2\pi]$, à l'aide de `odeint`, à l'aide de la méthode d'Euler, et de les comparer à la solution théorique qui pour cette C.I. vaut $y(t) = \dfrac{1}{\cos(t) + 7/3}$.

```
pl.figure(" y'(t)=y(t)^2 sin(t)")
pl.clf()
def F(y,t):
    return (y**2)*pl.sin(t)
y0=0.3
N=100
t=pl.linspace(0,2*pl.pi,N)
y=integrate.odeint(F,y0,t)
pl.plot(t,y,label="odeint")
z=1/(pl.cos(t)+7/3)
pl.plot(t,z,label="sol.exacte")
u=[y0]
pas=2*pl.pi/(N-1)
for i in range(len(t)-1):
    u.append(u[-1]+pas*(u[-1]**2)*pl.sin(t[i]))
pl.plot(t,u,label="Euler")
pl.legend()
pl.show()
```

3 Cas des équations du second ordre : comment on se ramène à une E.D. vectorielle du premier ordre

3.1 La méthode illustrée sur une E.D. Linéaire

Motivation : On se donne une EDL du second ordre $y''(t) = a(t)y'(t) + b(t)y(t) + c(t)$. On va la ramener à une E.D. du premier ordre mais à inconnue une *fonction à valeurs vectorielles*.

Intérêt ici : On pourra lui appliquer alors la commande `odeint` qui, suivant la philosophie de Pylab, s'applique aussi à des fonctions *vectorielles*.

Intérêt mathématique plus général : *On peut ramener toute la théorie des ED d'ordre quelconque à la théorie des E.D. d'ordre 1 dans le cadre vectoriel (cf. cours de maths de 2ème année).*

Définition de la dérivée d'une fonction de $\mathbb{R}$ dans $\mathbb{R}^2$: *coordonnées par coordonnées*
Si $f : t \mapsto (f_1(t), f_2(t))$, on définit $f'(t) = (f_1'(t), f_2'(t))$.

Retour à notre E.D. du second ordre :

On considère toujours l'E.D. $y''(t) = a(t)y'(t) + b(t)y(t) + c(t)$.
Idée : on pose $Y(t) = (y(t), y'(t))$. Avec la déf. précédente de la dérivée : $Y'(t) = (y'(t), y''(t))$, l'E.D. initiale est équivalente au système $\begin{cases} y'(t) = y'(t) \text{ (oui c'est trivial)} \\ y''(t) = a(t)y'(t) + b(t)y(t) + c(t). \end{cases}$

L'E.D. initiale peut donc s'écrire comme une équation du premier ordre pour la fonction $t \mapsto Y(t)$ sous la forme :

$$Y'(t) = F(Y(t), t),$$

où en notant $t \in \mathbb{R}$ et $Y = (Y_1, Y_2\ ) \in \mathbb{R}^2$, $F : (Y, t) \mapsto (Y_2, a(t)Y_2 + b(t)Y_1 + c(t))$.

3.2 Exemple de l'O.H avec `scipy`

On considère l'E.D. $y''(t) + \omega_0^2 y(t) = 0$. En posant $Y(t) = (y(t), y'(t))$, on peut la réécrire sous la forme $Y'(t) = F(Y(t), t)$ où pour $t \in \mathbb{R}$ et $Y = (Y_1, Y_2) \in \mathbb{R}^2$:

$$F(Y, t) = (Y_2, -\omega_0^2.Y_1).$$

Application en Python : on veut tracer la solution pour $\omega_0 = 2$, au problème de Cauchy posé par cette E.D. $y''(t) + 4y(t) = 0$ avec $y(0) = 1$ et $y'(0) = 0$. On utilise le script suivant :

```
from scipy import integrate
omega0=2
def F(Y,t):
    return (Y[1],-omega0**2*Y[0])
Y0=(1,0)# vecteur C.I.
t=pl.linspace(0,10,100)
Y=integrate.odeint(F,Y0,t)
```

Qu'y-a-t-il dans le `Y` renvoyé par `odeint` ?
Réponse : un tableau de taille `len(t)`×2 i.e. à deux colonnes.
Sur chaque ligne i, on a $Y(t_i) = (y(t_i), y'(t_i))$.

En voici le début :

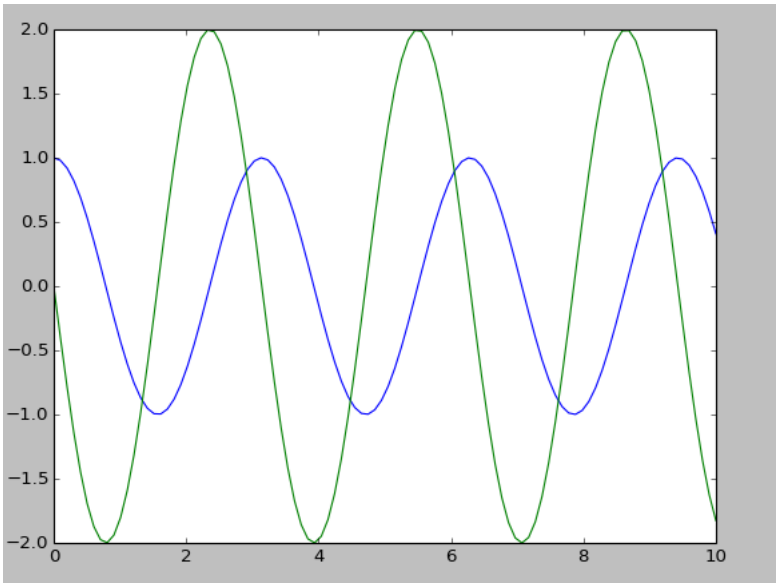
```
In [4]: Y
Out[4]:
array([[ 1.        ,  0.        ],
       [ 0.97966323, -0.4012977 ],
       [ 0.91948007, -0.78627321],
       [ 0.82189841, -1.13926819],
       [ 0.69088722, -1.4459251 ],
```

Remarque : on peut demander le nombre de lignes et de colonnes de `Y` avec `pl.shape(Y)`, qui renvoie ici $(100, 2)$.

3.3 Tracé des solutions de l'O.H.

Question : A partir du `Y` obtenu ci-dessus que se passe-t-il si on « fait » : `pl.plot(t,Y)` ?

Réponse en image :



Moralité : quand le second argument de `pl.plot` est un tableau à deux colonnes, il trace les deux courbes.

Question : Comment tracer une seule courbe, par exemple, seulement $t \mapsto y(t)$?

Réponse : En extrayant la première colonne du tableau précédent via `Y[:,0]`.

Explication : Pour un tableau numpy (pylab), on accède à l'entrée (i, j) (i-ème ligne, j-ème colonne) avec `Y[i,j]` (ou `Y[i][j]`).

Avec `Y[:,0]` on aura toutes les lignes mais seulement la colonne 0. De même avec `Y[1:4,0]` les lignes 1 à 3, mais seulement la colonne 0.

4 Application au portrait de phase de l'O.H.

4.1 Rappel sur l'obtention de l'intégrale première de l'énergie :

A partir de l'E.D. donnée par le P.F.D. pour un oscillateur harmonique :

$$my''(t) = -ky(t)$$

par multiplication des deux membres par $y'(t)$, on obtient

$$my'(t)y''(t) = -ky'(t)y(t)$$

qui équivaut à l'égalité :

$$\frac{d}{dt}(\frac{1}{2}my'(t)^2 + \frac{1}{2}ky(t)^2) = 0$$

et conduit à la conservation de l'énergie : il existe une constante E telle que pour tout t

$$\frac{1}{2}my'(t)^2 + \frac{1}{2}ky(t)^2 = E \quad (\dagger)$$

4.2 Une courbe dans l'espace des phases

Par déf. la courbe dans l'espace des phases correspondant à une solution de l'équation de l'O.H. est *la courbe paramétrée* $t \mapsto (y(t), y'(t))$. Avec le résultat du § 3.2, son tracé est immédiat :

```
pl.figure("Courbe dans l'espace des phases")
pl.clf()
pl.plot(Y[:,0],Y[:,1])
pl.show()
```

On obtient bien des ellipses comme prévus par $(\dagger)$. Mieux (cf. cours de physique) si on prend comme variables $(y(t), \frac{y'(t)}{\omega_0})$ on obtient un cercle puisque :

$$\frac{1}{2}ky^2 + \frac{1}{2}m(y')^2 = E \Leftrightarrow y^2 + (\frac{y'}{\omega_0})^2 = \frac{2E}{k} = C$$

ce qui se voit bien à condition de prendre des coordonnées en base orthonormée avec la commande `pl.axis("equal")`.

4.3 Tout un portrait ?

Pour avoir tout un portrait de phase, on doit *faire varier les C.I.* Cf. T.P.