Résolution numérique d'équations différentielles

6 mars 2018

Considérons l'équation différentielle suivante :

$$\begin{cases} y'(t) = 5y^2(t) - y(t) \times (1 + t^3) \\ y(0) = 1 \end{cases}$$

C'est une équation différentielle d'ordre 1, mais elle n'est pas linéaire. Nous ne savons pas la résoudre de manière exacte. Nous allons toutefois pouvoir la résoudre numériquement...

Remarquons déjà que cette équation peut s'écrire sous la forme $y'(t) = F(y(t), t)$ avec $F : \mathbb{R}^2 \to \mathbb{R}$ la fonction définie par $F(a, b) = 5a^2 - a(1 + b^3)$.

Plus généralement, étant donné $F : \mathbb{R}^2 \to \mathbb{R}$ (ou définie sur une partie de $\mathbb{R}^2$), le théorème de Cauchy-Lipschitz assure que sous des conditions raisonnables, il existe une unique application y de classe C^1 sur $[t_0, t_f]$ (t_f comme «temps final») vérifiant le problème de Cauchy :

$$\begin{cases} y'(t) = F(y(t), t) \\ y(t_0) = y_0 \end{cases}$$

Remarque : les ED linéaires d'ordre 1 peuvent bien sûr s'écrire sous la forme $y'(t) = F(y(t), t)$. Donnons quelques exemples :

- $y'(t) = (1 + t^2)y(t)$ s'écrit $y'(t) = F(y(t), t)$ avec $F(a, b) = (1 + b^2)a$.
- $y'(t) = y(t)$ s'écrit $y'(t) = F(y(t), t)$ avec $F(a, b) = a$.

L'objet des *schémas numériques* est d'obtenir des approximations de ces solutions dont la théorie donne l'existence mais ne dit pas comment les obtenir. Elles consistent en général à approximer la solution y en un certain nombre de points répartis sur $[t_0, t_f]$.

1 La méthode d'Euler

1.1 Le principe de la méthode

L'idée principale est que «localement la courbe de la fonction y ressemble à sa tangente». Ainsi si h est proche de 0, on a

$$y(t_0 + h) \sim y(t_0) + hy'(t_0) = y(t_0) + hF(y(t_0), t_0).$$

On peut donc approcher $y(t_0 + h)$ par la quantité $y(t_0) + hF(y(t_0), t_0)$.

On découpe ainsi l'intervalle de temps $[t_0, t_f]$ en n segments de même longueur $h = \frac{t_f - t_0}{n}$ (on dit que h est le pas). On dispose ainsi de $n+1$ temps $t_k = a + kh$ pour $k \in \{0, \dots, n\}$. On va alors approximer la solution y à l'instant t_k par le nombre y_k défini par la relation de récurrence :

$$y_{k+1} = y_k + hF(y_k, t_k).$$

On initialise enfin avec la condition initiale $y_0 = y(t_0)$.

```
def euler(F, t0,tf,y0, n):
    """Données:
        F(y,t) une fonction
        t0,t1 deux réels avec t0 < t1
        y0 un réel
        n un entier
        Résultat: le tuple constitué de la liste des temps [t0,...,tn] et la liste des (
        qui constituent une approximation de la solution y sur [t0,tf]
        de l'ED y'=F(y,t) avec la condition initiale y(t0) = y0
        """
    h = (tf-t0)/n
    y = y0
    t = t0
    Y = [y0]
    T = [t0]
    for k in range(n): # n itérations donc n+1 points
        y =  y + h*F(y,t)
        t = t + h
        Y.append(y)
        T.append(t)
    return T,Y
```

La quantité $h = \frac{t_f - t_0}{n}$ est appelé le pas. Plus le pas est petit, meilleure sera l'approximation.

Remarque : un autre point de vue équivalent et en lien avec les méthodes numériques d'intégration pourrait être :

$$y(t_{k+1}) - y(t_k) = \int_{t_k}^{t_{k+1}} y'(u)\mathrm{d}u = \int_{t_k}^{t_{k+1}} F(y(u), u)\mathrm{d}u \approx hF(y(t_k), t_k).$$

Cela revient à estimer l'intégrale $I = \int_{t_k}^{t_{k+1}} F(y(u), u)\mathrm{d}u$ par la méthode des rectangles à gauche sur $[t_k, t_{k+1}]$.

1.2 Mise en oeuvre de la méthode d'Euler

Appliquons notre fonction `euler` avec le cas d'école $y' = y$ et $y(0) = 1$. Nous la résolvons sur $[0, 1]$.

```
def F(y,t):
    return y

y0 = 1
n = 5
T5,Y5 = euler(F,0,1,y0,n)
```

Nous avons pris ici seulement $n = 5$.

```
In [74]: T5
Out[74]: [0, 0.2, 0.4, 0.6000000000000001, 0.8, 1.0]

In [75]: Y5
Out[75]: [1, 1.2, 1.44, 1.728, 2.0736, 2.48832]
```

La dernière valeur de Y5 auquel on peut accéder par `Y5[-1]` est une approximation de $\exp(1) = \mathrm{e}$.

Observons l'évolution de l'erreur commise à l'instant 1 lorsque l'on augmente n :

```
for n in [10,100,1000]:
    T,Y = euler(F, 0, 1, y0, n)
    erreur = abs(Y[-1] -np.exp(1))
    print(erreur)
```

Les valeurs affichées sont $0.124539368359,\ 0.0134679990375,\ 0.00135789622315$.

On observe que si n est multiplié par 10, donc le pas divisé par 10, l'erreur semble elle aussi divisée par 10. Cela semble indiquer que la méthode d'Euler est une méthode d'ordre 1. On peut démontrer que c'est effectivement le cas.

Remarque : dans notre cas d'école, on a $y_{k+1} = y_k + hy_k = (1+h)y_k$ d'où $y_n = y_0(1+h)^n = (1+h)^n$. Si l'on découpe $[0,1]$ en n intervalles, $h = \frac{1}{n}$ et donc $y_n = (1+\frac{1}{n})^n$ est une approximation de exp en 1, c'est-à-dire du nombre e. On peut effectivement montrer que

$$(1+\frac{1}{n})^n - \mathrm{e} = \frac{-\mathrm{e}}{2n} + o(\frac{1}{n}).$$

ce qui corrobore que la méthode est d'ordre 1.

Voici les courbes obtenus pour différentes valeurs de n.

2 Pour la culture : d'autres méthodes numériques

D'autres méthodes numériques très classiques améliorent la méthode d'Euler mais reposent sur le même principe : ayant un pas fixé h, on construit encore une suite :

$$\left\{ \begin{array}{rcl} t_{k+1} & = & t_k + h \\ y_{k+1} & = & y_k + hm \end{array} \right.$$

où le nombre m peut être assimilé à une *pente moyenne* sur l'intervalle $[t_k, t_{k+1}]$.

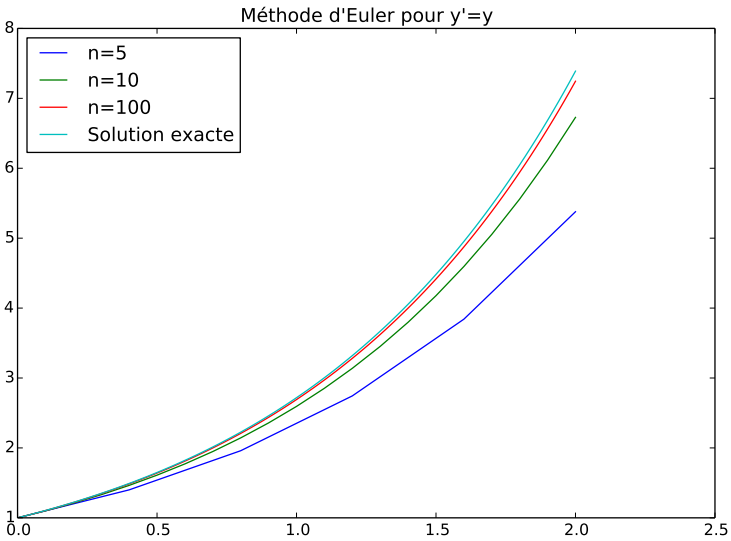

FIGURE 1 – Approximation de exp par la méthode d'Euler

- pour la méthode du point milieu, on prend pour m la valeur estimé de $u \mapsto F(y(u), u)$ au milieu de $[t_k, t_{k+1}]$, c'est-à-dire

$$m = F(y_k + \frac{h}{2}F(y_k, t_k), t_k + \frac{h}{2}).$$

  Remarques :

  – cela revient à estimer l'intégrale I ci-dessus par la méthode du point milieu sur $[t_k, t_{k+1}]$.
  – on peut montrer que c'est une méthode d'ordre deux.

- pour la méthode de Heun, m est la moyenne de la dérivée en t_k et de la dérivée en t_{k+1} estimée par Euler, c'est-à-dire :

$$m = \frac{F(y_k, t_k) + F\left(y_k + hF(y_k, t_k), t_{k+1}\right)}{2}.$$

  Remarques :

  – cela revient à estimer l'intégrale I ci-dessus par la méthode des trapèzes sur $[t_k, t_{k+1}]$.
  – on peut montrer que c'est une méthode d'ordre deux.

- pour la méthode RK4 dite de «Runge-Kutta d'ordre 4», m est la moyenne pondérée de quatre pentes. C'est la plus performante des quatre méthodes citées, elle est d'ordre quatre.

3 Et avec Python ?

Il faut utiliser la fonction `odeint` de Python de la librairie `scipy.integrate`.

```
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt
```

Ne pas hésiter à regarder la notice de la fonction avec `help(odeint)`.

Elle permet de résoudre des ED du type $y'(t) = F(y(t), t)$ avec $y(t_0) = y_0$. Elle prend en argument la fonction F, la condition initiale y_0 et une liste de temps commençant à t_0. Par exemple, pour résoudre $y' = y$ sur $[0, 1]$ avec $y(0) = 1$. On pourra écrire le script suivant :

```
def F(y,t):
    return y

temps = np.linspace(0, 1, 10)
Y = odeint(F, 1, temps)

In [14]:Y
Out[14]:
array([[ 1.        ],
       [ 1.11751906],
       [ 1.24884886],
       [ 1.39561243],
       [ 1.55962349],
       [ 1.742909  ],
       [ 1.94773405],
       [ 2.17663003],
       [ 2.43242551],
       [ 2.7182819 ]])
```

On peut tracer la solution obtenue :

```
plt.plot(temps,Y)
plt.show()
```

On peut aussi résoudre des systèmes différentiels. Traitons l'exemple suivant, un modèle «proie-prédateur» où $x(t)$ (resp. $y(t)$) représente la quantité de renards (resp. de lapins) à l'instant t. On prend comme conditions initiales $x_0 = 6$ et $y_0 = 4$ et on fait une étude sur une durée de 10 ans.

$$\left\{ \begin{array}{rcl} x'(t) & = & x(t)(3 - 2y(t)) \\ y'(t) & = & -y(t)(4 - x(t)) \end{array} \right. \qquad (E)$$

On «vectorialise» notre ED : on pose $X(t) = (x(t), y(t)$, alors

$$X'(t) = (x'(t), y'(t)) = (x(t)(3 - 2y(t)), -y(t)(4 - x(t))) .$$

Le système linéaire (E) s'écrit donc sous la forme $X'(t) = F(X(t), t)$ avec $F : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2$ définie par :

$$F((x, y), t) = (x(3 - 2y), -y(4 - x)).$$

```
def F(X,t):
    x,y = X[0],X[1] # ou bien x,y = X
    return (x*(3-2*y),y*(x-4))

temps = np.linspace(0,10, 1000)
sol = odeint(F,[6,4], temps)
```

La variable `sol` est un tableau numpy de 1000 lignes et deux colonnes

```
In [46]: type(sol)
Out[46]: numpy.ndarray

In [47]: sol.shape
Out[47]: (1000, 2)
```

La première colonne correspond aux valeurs de $x(t)$ et et la deuxième aux valeurs de $y(t)$.

On récupère ces deux colonnes avec du slicing.

```
lapins = sol[ :, 0]
renards = sol[ :, 1]
```

Il n'y a plus qu'à tracer.

```
# Evolution des populations
plt.plot(temps,lapins)
plt.plot(temps,renards)
plt.show()


# Portrait de phase
plt.plot(lapins, renards)
plt.show()
```

4 Et pour les ED d'ordre supérieur ?

Prenons l'exemple du pendule qui conduit à une équation différentielle d'ordre deux scalaire (solution à valeurs dans $\mathbb{R}$) :

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

L'idée est que l'on peut transformer cette ED d'ordre deux, en un système différentiel d'ordre 1 ou une équation différentielle d'ordre 1 vectorielle (à valeurs dans $\mathbb{R}^2$). En effet, on pose $X(t) = (\theta(t), \theta'(t))$. Alors $X'(t) = (\theta'(t), \theta''(t)) = (\theta'(t), -\sin\theta(t))$. Ainsi l'ED est équivalente à $X'(t) = F(X(t), t)$ avec

$$F((x, y), t) = (y, -\sin x).$$

On résoud l'équation sur $[0, 6\pi]$ et on prend comme conditions initiales $\theta(0) = 0$ et $\theta'(0) = 0.5$.

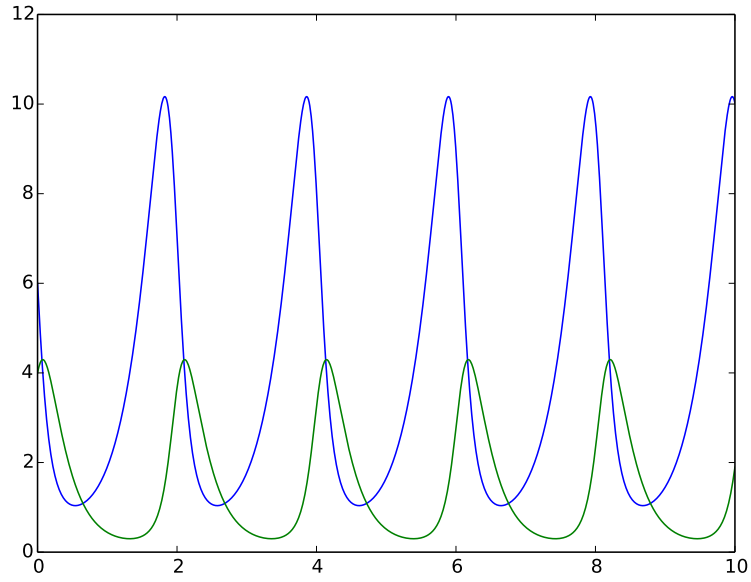

FIGURE 2 – population de lapins et renards en fonction du temps

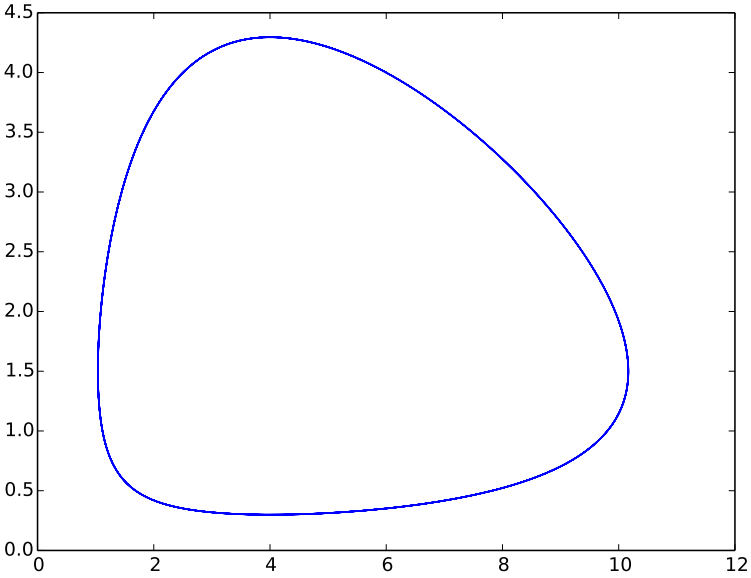

FIGURE 3 – un portrait de phase

4.1 Résolution avec odeint

```
def F(Y,t):
    """Données: t un flottant, Y un tableau de deux flottants"
     Résulat: un tableau de deux flottants
     """
    theta,thetap = Y
```

```
    return np.array([thetap, - np.sin(theta)])

temps = np.linspace(0, 6*np.pi, 100)
sol = odeint(F, [0, 0.5], temps)
theta , thetap = sol[:, 0], sol[ :, 1]
plt.plot(temps, theta)
plt.show()
```

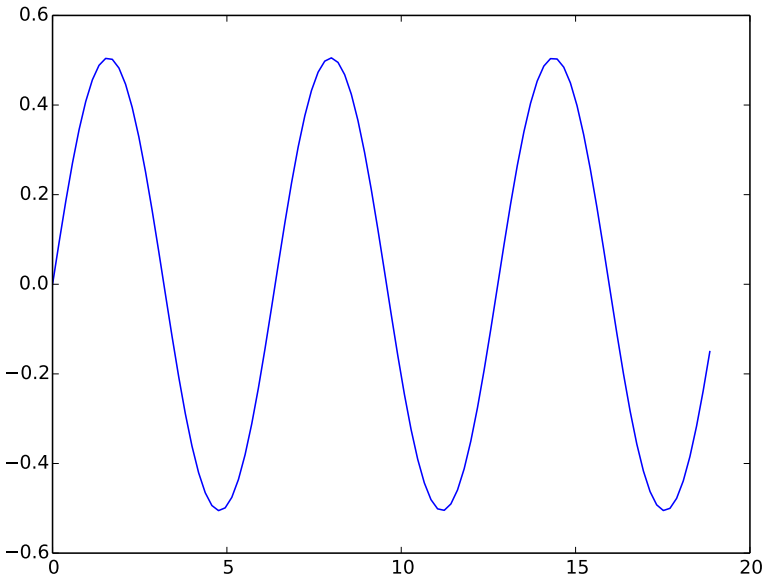

FIGURE 4 – le pendule non amorti

4.2 Résolution avec la méthode d'Euler

Le code `euler` que l'on a écrit pour les ED scalaires peut être recopié mot pour mot pour une ED vectorielle à condition de comprendre que la ligne de code `y = y + h*F(y,t)` doit être une combinaison linéaire de vecteurs. Il faut pour cela que la variable y soit un tableau numpy (de type `ndarray`). Si y est une liste par exemple $[1, 2]$, alors $y + y$ ne vaut pas $[2, 4]$ comme on pourrait l'espérer mais $[1, 2, 1, 2]$ (l'opérateur + l'opérateur de concaténation pour les listes). En revanche si `y = np.array[1,2]`, les choses se passent comme on le souhaite :

```
In [29]: y =np.array([1,2])
In [30]: y+y
Out[30]: array([2, 4])
```

On reprend la résolution de l'ED du pendule avec Euler :

```
y0 = np.array([0, 0.5])
sol = euler(F,0, 6*np.pi,y0,1000)
```

```
temps, Y = sol[0], sol[1]  # attention Y est une liste de array
Y = np.array(Y)
theta, thetap = Y[:, 0], Y[ :, 1]

plt.plot(temps, theta)
plt.show()
```