Chapitre 6

Résolution numérique d'une équation différentielle

Considérons une équation différentielle *scalaire ou vectorielle* la forme : $x' = f(x,t)$ où f est une fonction définie sur une partie $\mathcal{U}$ de $\mathbb{R}^p \times \mathbb{R}$, à valeurs dans $\mathbb{R}^p$.
Pour assurer l'unicité de la solution, on adjoint à cette équation une condition initiale sous la forme d'un couple $(x_0, t_0) \in \mathcal{U}$ et chercher à résoudre le problème de Cauchy suivant :

$$\begin{cases} x' = f(x,t) \\ x(t_0) = x_0 \end{cases}$$

Sous certaines conditions sur f que nous ne détaillerons pas, ce problème admet une unique solution, que nous allons chercher à déterminer numériquement.

1. La méthode d'Euler

Elle consiste à subdiviser l'intervalle de temps $[t_0, t_0 + \mathrm{T}]$ en $n+1$ points $t_0 < t_1 < \cdots < t_n = t_0 + \mathrm{T}$ puis à utiliser la méthode des rectangles gauche pour approcher la relation :

$$x(t_{k+1}) - x(t_k) = \int_{t_k}^{t_{k+1}} x'(t)\,\mathrm{d}t \qquad \text{par} \qquad x(t_{k+1}) - x(t_k) \approx (t_{k+1} - t_k)x'(t_k)$$

autrement dit : $x(t_{k+1}) \approx x(t_k) + (t_{k+1} - t_k) f(x(t_k), t_k)$.
En posant $h_k = t_{k+1} - t_k$, ceci conduit à définir une suite de valeurs $x_0, x_1, \ldots, x_n$ à partir de la condition initiale x_0 et de la relation de récurrence :

$$\forall k \in [\![0, n-1]\!], \quad \boxed{x_{k+1} = x_k + h_k f(x_k, t_k)}$$

On observera qu'en général, seul le premier point x_0 de cette méthode est une valeur exacte ; les autres points sont calculés à partir de l'approximation précédente, ce qui peut conduire la valeur calculée x_k à s'écarter de plus en plus de la valeur exacte $x(t_k)$. D'autres méthodes, plus efficaces mais hors-programme, permettent de corriger en grande partie ce problème.
En ce qui concerne la méthode d'Euler, on se convaincra aisément que lorsque la subdivision s'affine la précision en général s'améliore (sans pour autant faire disparaître ce phénomène de divergence).

Remarque. Pour discrétiser l'intervalle $[t_0, t_0 + \mathrm{T}]$ on utilise en général une subdivision de pas régulier (pour lequel la quantité h_k est constante) ; cependant de meilleurs résultats sont obtenus en adaptant le pas à la fonction f : si $f(x_k, t_k)$ est faible alors x varie peu et on peut utiliser un pas plus grand. On contraire, on réduira le pas lorsque la valeur de $f(x_k, t_k)$ augmente. Dans ce cas, on parle de méthode à *pas adaptatif* ; une telle méthode est présentée dans l'exercice 4.

Exercice 1.

a) Rédiger une fonction `euler(f, x0, t)` qui prend pour arguments la fonction f, la valeur x_0 et un tableau `t` contenant les valeurs $[t_0, t_1, \ldots, t_n]$ et qui renvoie le tableau $[x_0, x_1, \ldots, x_n]$.

b) Rédiger une fonction `euler(f, x0, t0, T, n)` qui prend pour arguments la fonction f, les valeurs x_0 et t_0, la durée T et le nombre de points n de la subdivision de pas régulier et qui renvoie le tableau $[x_0, x_1, \ldots, x_n]$.

Remarque. La fonction `odeint` du module `scipy.integrate` suit la même syntaxe que la fonction `euler` écrite à la première question de cet exercice, et renvoie un tableau de type `numpy.array`.

2. La méthode de Heun (hors programme)

À l'instar de la méthode d'Euler, cette méthode de résolution numérique de l'équation différentielle $x' = f(x,t)$ consiste à subdiviser l'intervalle de temps $[t_0, t_0 + \mathrm{T}]$ en $n+1$ points $t_0 < t_1 < \cdots < t_n = t_0 + \mathrm{T}$ puis à approcher la quantité $x(t_{k+1}) - x(t_k) = \displaystyle\int_{t_k}^{t_{k+1}} x'(t)\,\mathrm{d}t$ en utilisant non pas la méthode des rectangles mais la méthode des trapèzes :

$$x(t_{k+1}) - x(t_k) \approx (t_{k+1} - t_k)\frac{x'(t_k) + x'(t_{k+1})}{2} = (t_{k+1} - t_k)\frac{f(x(t_k), t_k) + f(x(t_{k+1}), t_{k+1})}{2}.$$

Dans l'idéal il faudrait donc définir x_{k+1} à partir de x_k de la façon suivante :

$$x_{k+1} = x_k + h_k \frac{f(x_k, t_k) + f(x_{k+1}, t_{k+1})}{2}$$

mais on constate que cette formule est inadéquate puisque x_{k+1} est présent dans la partie droite de l'expression. C'est la raison pour laquelle on remplace ce dernier terme par son approximation d'Euler $x_k + h_k f(x_k, t_k)$ ce qui conduit au schéma d'approximation suivant :

$$x_{k+1} = x_k + h_k \frac{f(x_k, t_k) + f(x_k + h_k f(x_k, t_k), t_{k+1})}{2}.$$

Exercice 2. Rédiger une fonction `heun(f, x0, t)` qui prend pour arguments la fonction f, la valeur x_0 et un tableau `t` contenant les valeurs $[t_0, t_1, \ldots, t_n]$ et qui renvoie le tableau $[x_0, x_1, \ldots, x_n]$ correspondant à la méthode d'approximation de Heun.

3. La méthode d'Euler pour une équation d'ordre 2

Considérons maintenant un problème de Cauchy d'ordre 2, c'est-à-dire de la forme

$$\begin{cases} x'' = f(x, x', t) \\ x(t_0) = x_0 \\ x'(t_0) = y_0 \end{cases}$$

La méthode consiste à se ramener à un système différentiel d'ordre 1 en considérant le vecteur $\mathrm{X} = (x, x')$. Ce dernier est en effet solution du système différentiel :

$$\begin{cases} x' = y \\ y' = f(x, y, t) \end{cases} \quad \text{soit} \quad \mathrm{X}' = \mathrm{F}(\mathrm{X}, t) \quad \text{avec} \quad \mathrm{F}\Big((x, y), t\Big) = \Big(y, f(x, y, t)\Big).$$

Il s'agit donc ici d'itérer deux suites (x_n) et (y_n) définies par les conditions initiales (x_0, y_0) et les relations :

$$\boxed{\begin{aligned} x_{k+1} &= x_k + h_k y_k \\ y_{k+1} &= y_k + h_k f(x_k, y_k, t_k) \end{aligned}}$$

Les termes $(x_0, \ldots, x_n)$ fournissent une approximation de la fonction x aux temps $(t_0, \ldots, t_n)$ et $(y_0, \ldots, y_n)$ une approximation de la fonction dérivée x' à ces mêmes dates.

Exercice 3. Rédiger une fonction `euler2(f, x0, y0, t)` qui prend pour arguments la fonction f, les valeurs x_0 et y_0 et un tableau `t` contenant les valeurs $[t_0, t_1, \ldots, t_n]$ et qui renvoie le couple de tableaux $[x_0, x_1, \ldots, x_n]$, $[y_0, y_1, \ldots, y_n]$.

Remarque. La fonction `odeint` du module `numpy.integrate` ne s'applique qu'aux équations différentielles du premier ordre, mais en revanche elle s'applique aux fonctions vectorielles. Il faut donc appliquer à chaque fois la démarche présentée ci dessus.

Considérons par exemple le problème suivant, modélisant un pendule pesant dans le champ de pesanteur soumis à une force de frottement fluide proportionnelle à la vitesse :

$$\begin{cases} x'' = -\sin(x) - \dfrac{1}{4}x' \\ x(0) = 0 \\ x'(0) = 2 \end{cases}$$

Pour résoudre ce système avec `odeint` on « vectorialise » ce problème en posant $X = (x, x')$; on obtient le système d'ordre 1 suivant :

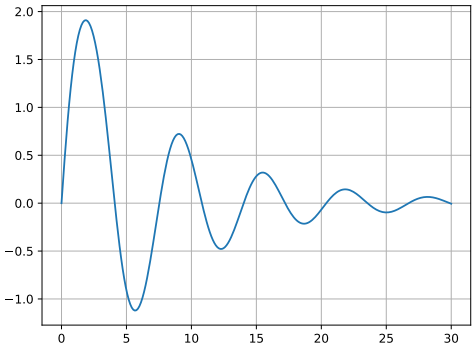
$$\begin{cases} X' = F(X, t) \\ X(0) = (0, 2) \end{cases} \qquad \text{avec} \qquad F\Big((x, x'), t\Big) = \Big(x', -\sin(x) - x'/4\Big)$$

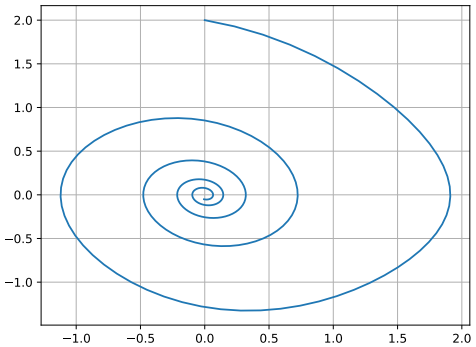
ce qui conduit au code suivant :

```
import numpy as np
from scipy.integrate import odeint

def F(X, t):
    (x, dx) = X
    return (dx, -np.sin(x) - dx / 4)

t = np.linspace(0, 30, 256)  # l'intervalle temporel d'étude est [0, 30]
X = odeint(F, [0, 2], t)
```

Ici, la variable `X` est un tableau numpy contenant les valeurs vectorielles $\Big[[x_0, x'_0], [x_1, x'_1], \ldots, [x_{n-1}, x'_{n-1}]\Big]$ (c'est donc un tableau de dimension $n \times 2$). La coupe `X[:, 0]` contient donc les valeurs de x et la coupe `X[:, 1]` les valeurs de x', ce qui permet de tracer le graphe (t, x) mais aussi le portrait des phase (x, x').

```
import matplotlib.pyplot as plt

plt.plot(t, X[:, 0])
plt.show()

plt.plot(X[:, 0], X[:, 1])
plt.show()
```





4. Exercices

Exercice 4 **Le schéma d'Euler-Richardson**

Le schéma d'intégration d'Euler-Richardson n'utilise pas une discrétisation du temps fixée à l'avance mais adapte son pas h en fonction de la situation : pour cela, on compare les pentes $p = f(x_k, t_k)$ et $p' = f(x_k + ph/2, t_k + h/2)$. Si celles-ci sont trop éloignées, on réduit h ; à l'inverse si elles sont très proches, le pas h est augmenté.

Plus précisément, on fixe un seuil de précision $\epsilon > 0$ puis à chaque étape on détermine $e = \dfrac{h}{2}|p' - p|$.

– si $e > \epsilon$, on réduit le pas avec $h \leftarrow 0,9h\sqrt{\epsilon/e}$ puis on revient au calcul de p et p' ;
– si $e < \epsilon$, on pose $t_{k+1} = t_k + h$, $x_{k+1} = x_k + p'h$ puis on augmente le pas avec $h \leftarrow 0,9h\sqrt{\epsilon/e}$.

On procède ainsi jusqu'à atteindre la date $t_0 + \mathrm{T}$.
Rédiger une fonction `adaptatif(f, x0, t0, T, h, epsilon)` qui prend pour arguments la fonction f, les valeurs initiales x_0 et t_0 la durée de l'expérience T, le pas initial h et la précision ϵ, et qui renvoie les tableaux des valeurs $[t_0, t_1, \ldots, t_n]$ et $[x_0, \ldots, x_n]$ correspondant au schéma d'Euler-Richardson.

Exercice 5 Le schéma de Verlet

L'étude de systèmes dynamiques conservatifs demande de résoudre des équations différentielles de la forme $x'' = f(x)$ dans lesquelles les quantités t et x' sont absentes.
Pour un tel système la quantité $\mathrm{E}(t) = \frac{1}{2}x'(t)^2 - \int_{x_0}^{x(t)} f(u)\,\mathrm{d}u$ est constante, ce qui traduit pour le physicien la conservation de l'énergie du système.
Si on pose $\mathrm{E}_k = \frac{1}{2}y_k^2 - \int_0^{x_k} f(u)\,\mathrm{d}u$ il est donc souhaitable qu'un schéma numérique d'approximation préserve le plus possible le caractère constant de cette quantité, autrement dit que la quantité

$$\Delta_k = \mathrm{E}_{k+1} - \mathrm{E}_k = \frac{1}{2}y_{k+1}^2 - \frac{1}{2}y_k^2 - \int_{x_k}^{x_{k+1}} f(u)\,\mathrm{d}u \approx \frac{1}{2}y_{k+1}^2 - \frac{1}{2}y_k^2 - (x_{k+1} - x_k)f(x_k)$$

soit la plus petite possible. Or dans le cas de la méthode d'Euler, on a :

$$\Delta_k = \frac{1}{2}\Big(y_k + h_k f(x_k)\Big)^2 - \frac{1}{2}y_k^2 - (x_{k+1} - x_k)f(x_k) = \frac{1}{2}h_k^2 f(x_k)^2 > 0.$$

Appliquer la méthode d'Euler à un système dynamique conservatif conduit donc toujours à un accroissement (non réaliste) de l'énergie du système. C'est la raison pour laquelle on utilise un autre schéma, le *schéma de Verlet*, qui utilise les formules suivantes :

$$\begin{cases} x_{k+1} = x_k + h_k y_k + \dfrac{h_k^2}{2} f(x_k) \\ y_{k+1} = y_k + \dfrac{h_k}{2}\Big(f(x_k) + f(x_{k+1})\Big) \end{cases}$$

Rédiger la fonction `verlet(f, x0, y0, t)` correspondante, qui prend pour arguments la fonction f, les valeurs x_0 et y_0 et un tableau `t` contenant les valeurs $[t_0, t_1, \ldots, t_n]$ et qui renvoie le couple de tableaux $[x_0, x_1, \ldots, x_n]$, $[y_0, y_1, \ldots, y_n]$.