TP 4 : Attracteur de Lorenz

Cours de modélisation numérique

Vendredi 17 mars 2017

Rappel théorique : approximation numérique d'équations différentielles ordinaires

En général il n'existe pas de solutions analytiques pour les équations différentielles ordinaire d'un interêt pratique, c'est pourquoi on est obligé d'utiliser des méthodes numériques pour approximer la solution. Nous présentons ici deux familles de méthodes numériques : les **méthodes de Runge-Kutta** et les **méthodes d'Adams implicites** qui permettent d'approximer les solutions d'équations différentielles ordinaires :

$$\frac{dy}{dt} = f(t, y(t)), \quad y(t_0) = y_0 \tag{1}$$

où $y(t) = (y_1(t), \ldots, y_n(t))^T$ et $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$.

Les méthodes de Runge-Kutta

En analyse numérique les méthodes de Runge-Kutta (RK) sont une famille importante de méthodes permettant d'approximer les solutions d'équations différentielles ordinaires. Pour approximer la solution de l'éq. (1) sur l'intervalle $[t_0, t_{\text{end}}]$ avec ces méthodes, on commence par subdiviser cet intervalle en k sous-intervalles de tailles h. Les intervalles sont donnés par :

$$t_0,\ t_1 = t_0 + h,\ t_2 = t_1 + h, \ldots,\ t_k = t_{k-1} + h,$$

où $t_{end} = t_k$. On calcule ensuite une approximation $y_i \approx y(t_i)$ par une formule de la forme :

$$y_{i+1} = y_i + h\varphi(h, t_i, y_i)$$

Pour dériver des méthodes numériques, l'idée des méthodes de RK est d'intégrer éq. (1) :

$$y(t_{i+1}) = y_i + \int_{t_i}^{t_{i+1}} f(\tau, y(\tau))\ d\tau \tag{2}$$

et d'approximer l'intégrale par une formule de quadrature. Par exemple, en remplacant l'intégrale par $hf(t_i, y_i)$, on obtient la **méthode d'Euler** :

$$y_{i+1} = y_i + hf(t_i, y_i) \tag{3}$$

Cette méthode fait partie de la famille des méthodes de RK dites à s étages. La forme générale d'une méthode de RK à s étages est :

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i, \tag{4}$$

où h est, comme précédemment, le pas de temps discret : en une itération, le système avance d'un temps t au temps $t+h$. La valeur des k_i est

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + c_2 h, y_n + a_{21} h k_1) \\ &\vdots \\ k_s &= f(t_n + c_s h, y_n + a_{s1} h k_1 + a_{s2} h k_2 + \cdots + a_{s,s-1} h k_{s-1}). \end{aligned}$$

et où les a_{ij}, b_i et c_i sont des coefficients propre à la méthode. Par exemple la méthode d'Euler est la formule de RK à un étage ($s = 1$) dont les coefficients valent $c_1 = 0$ et $b_1 = 1$.

Les méthodes d'Adams implicites

Les méthodes d'Adams implicites font partie de la famille des méthodes multipas. Contrairement aux méthodes de Runge-Kutta qui n'utilisent que le pas précédent pour calculer le suivant, ces méthodes utilisent l'information des $k+1$ pas successifs $y_{t+1}, y_t, y_{t-1}, \ldots, y_{t-k+1}$ pour obtenir une approximation précise de y_{t+1}.

Pour dériver une méthode numérique, l'idée est la même que celle utilisé pour les méthode de RK, sauf qu'en place d'utiliser une méthode de quadrature pour approximer l'intégrale dans l'éq. (2), on remplace $f(t, y(t))$ par le polynome $p(t)$ de degré k satisfaisant

$$p(t_j) = f(t_j, y_j)$$

pour $j = t+1, t, t-1, \ldots, t-k+1$. L'approximation est donc définie par

$$y(t_i + h) = y_i + \int_{t_i}^{t_i+h} p(\tau)\ d\tau \tag{5}$$

En utilisant la formule de Newton pour représenter le polynome $p(t)$, la forme générale d'une méthode d'Adams implicite est la suivante :

$$y_{t+1} = y_t + h \sum_{j=0}^{k} \gamma_j \nabla^j f(t_{t+1}, y_{t+1}), \tag{6}$$

où $\nabla^j f_{t+1}$ sont les différences finies régressives et γ_j sont des coefficients propres à la méthode.

Pour cette série d'exercices nous nous intéresserons à la formule du **trapèze implicite** ($k = 1$), c'est-à-dire que nous utiliserons uniquement l'information du pas précédent pour calculer l'approximation de y_{t+1} (comme avec les méthode de RK). Dans ce cas $\gamma_0 = 1$, $\gamma_1 = -\frac{1}{2}$, $\nabla^0 f_{t+1} = f_{t+1}$ et $\nabla^1 f_{t+1} = f_{t+1} - f_t$, ce qui donne :

$$y_{t+1} = y_t + \frac{h}{2}\big(f(t_{t+1}, y_{t+1}) + f(t_t, y_t)\big). \tag{7}$$

On constate que y_{t+1} est encore inconnu. Pour trouver y_{t+1} il faut donc résoudre une équation non-linéaire du genre $y_{t+1} = g(y_{t+1})$. Nous vous proposons d'utiliser la méthode du point fixe, qui consiste à appliquer l'itération suivante :

$$x_{i+1} = g(x_i), \text{ avec } i = 0, 1, 2, \ldots \tag{8}$$

qui produit une séquence $x_0, x_1, \ldots$ convergeant vers un point x tel que $g(x) = x$ si g est continue.

Dans le cas de cette série, vous cherchez à résoudre à chaque pas le problème suivant :

$$x = y_t + \frac{h}{2}\big(\psi(x) + \psi(y_t)\big)$$

où la fonction ψ est l'équation de Lorenz (éq. 9). En utilisant $x_0 = y_t$, les itérations de la méthode du point fixe auront donc la forme suivante :

$$\begin{aligned} x_1 &= y_t + \frac{h}{2}\big(\psi(x_0) + \psi(y_t)\big) \\ x_2 &= y_t + \frac{h}{2}\big(\psi(x_1) + \psi(y_t)\big) \\ &\vdots \\ x_n &= y_t + \frac{h}{2}\big(\psi(x_{n-1}) + \psi(y_t)\big). \end{aligned}$$

Commentaire sur le pas de temps h

Dans ce TP on fait évoluer les schémas numériques avec un pas de temps h fixe qui est imposé par l'utilisateur. Dans un contexte général, il pourrait être plus efficace et/ou plus précis de choisir un pas de temps adaptatif qui s'ajuste au cours de l'évolution du système pour garantir une précision optimale.

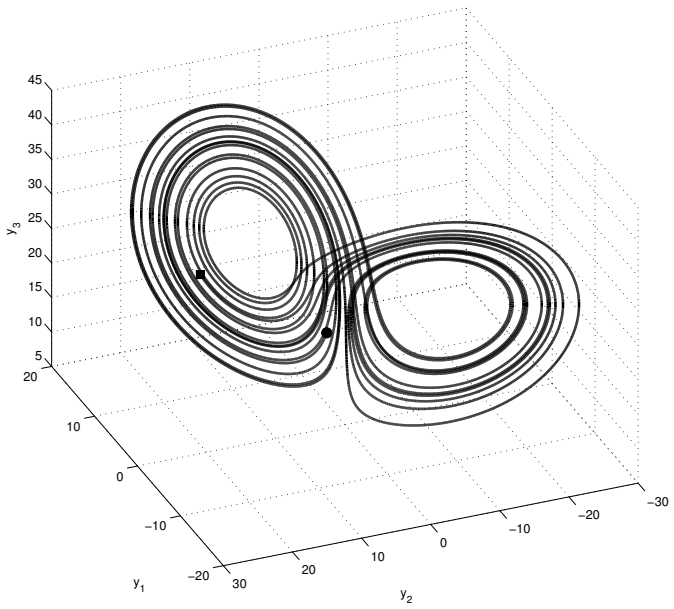
Equation de Lorenz

L'équation de Lorenz est un système d'équations différentielles ordinaires agissant sur une variable vectorielle à trois composantes $y(t) = \big(y_1(t), y_2(t), y_3(t)\big)$.

$$\begin{aligned} \tfrac{dy_1}{dt} &= \sigma(y_2 - y_1), \\ \tfrac{dy_2}{dt} &= y_1(\rho - y_3) - y_2, \\ \tfrac{dy_3}{dt} &= y_1 y_2 - \beta y_3, \end{aligned} \tag{9}$$

où les constantes sont définies par $\beta = 8/3$, $\sigma = 10$, $\rho = 28$.

Voici la trajectoire temporelle du point $\big(y_1(t), y_2(t), y_3(t)\big)$ à partir d'un point initial quelconque :



Quelques commentaires sur ce système :

— Ce système est un système dynamique chaotique. Si on perturbe la valeur de la condition initiale, cette perturbation croîtra de manière exponentielle au cours du temps.

— On peut voir que la trajectoire évolue sur une structure qui est essentiellement bidimensionelle ("le papillon"). En plus, la structure formée par le passage de la trajectoire est fractale : il existe une infinité d'anneaux blancs par lesquels la trajectoire ne passera jamais. Cette structure constitue l'attracteur de l'équation de Lorenz. Quelle que soit la condition initiale, la trajectoire du système finit par évoluer dans l'attracteur.

— A l'intérieur de l'attracteur, on ne trouve pas de points fixes ni de cycles limites stables.

Travail à faire

Solvers pour les équations differentielles ordinaires

Implémentez des solvers d'équations différentielles ordinaires pour la méthode d'Euler explicite et pour la méthode du trapèze implicite. Ces solvers avanceront par pas temporels h fixes : il faudra qu'on puisse leur fournir la valeur de h en paramètre. Faites en sorte que vos solvers marchent avec des valeurs y qui peuvent être des scalaires ou des vecteurs n-dimensionnels.

Les signatures proposées (mais pas imposées) sont les suivantes :

— `def euler( f, y0, h, t_end )` avec `f` la fonction à intégrer, `y0` la valeur initiale, `h` le pas de temps et `t_end` le temps final.

— `def trapeze( f, y0, h, t_end )` avec `f` la fonction à intégrer, `y0` la valeur initiale, `h` le pas de temps et `t_end` le temps final.

— `def lorrenz( y )` avec `y` le vecteur $y = (y_1, y_2, y_3)$ la variable sur laquelle le système d'équation agit.

— `def point_fixe( f, x )` avec `f` la fonction qui va être appliquée itérativement et `x` la valeur de x_0 (8).

La fonction `arange` de `numpy` vous permet de facilement créer des sous-intervalles. Par exemple avec un $t_0 = 0$, un $t_{end} = 4$ et un pas $h = 0.5$ on construit :

```
import numpy as np
np.arange( 0, 4, 0.5 )
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5])
```

qui contient toutes les bornes de vos sous-intervalles, à l'exception du t_{end} (connu). Votre programme pourrait avoir la forme suivate :

```
def euler(...):
  ...

def trapeze(...):
  ...

def lorenz(...):
  ...

def point_fixe(...):
  ...

y0 = ...

#méthode d'Euler
ye = euler( lorenz, y0, h, 1.6 )
plt.plot( ye[0], ye[1], ye[2], ... )
plt.show()

#méthode du trapèze
yt = trapeze( lorenz, y0, h, 1.6 )
plt.plot( yt[0], yt[1], yt[2], ... )
plt.show()
```

Notez que le fichier *example.py* disponible sur chamilo montre un exemple simple de création de courbes en 3D, pour présenter vos résultats.

Equation de Lorenz

Appliquez vos solvers à l'équation de Lorenz et reproduisez une figure affichant l'attracteur de Lorenz, comme celle qui est fournie dans cet énoncé.

Orbite périodique instable sur l'attracteur de Lorenz

L'attracteur de Lorenz contient des trajectoires périodiques sur lesquelles le système revient au point initial au bout d'un moment. Ces trajectoires sont instables, comme toute autre trajectoire de ce système : il suffit qu'on dévie un tout petit peu de la trajectoire, et on finit par s'en éloigner complètement. Le

point suivant se trouve (dans la limite de la précision numérique fournie) sur une trajectoire périodique de période $T = 1.5586522$:

$$y = (-0.9101673912,\ -1.922121396,\ 18.18952097).$$

Prenez ce point comme valeur initiale, et faites évoluer le système avec un pas de temps donné (par exemple $h = 0.001$). Combien de temps arrivez-vous à rester sur l'orbite périodique avec Euler ? Et avec la méthode du trapèze implicite ? Tracez les deux trajectoires sur une figure 3D pour les comparer.