Lycée Louis-Le-Grand, Paris
MPSI 4 – Informatique pour tous
N. Carré, A. Troesch

2020/2021

TP n° 10 : Équations différentielles

Exercice 1 – (Résolutions numériques d'équations différentielles)

On étudie dans cet exercice plusieurs méthodes de résolution numérique d'équations différentielles $y' = f(y,t)$, où f est une fonction de deux variables réelles à valeurs dans $\mathbb{R}$.

On testera les méthodes implémentées dans cet exercice avec les équations définies par $f_1(y,t) = y$ entre 0 et 2, et $f_2(y,t) = \sin(t)\sin(y)$ entre 0 et 50, avec la condition initiale $y(0) = 1$ dans les deux cas. On comparera les résultats obtenus avec le résultat théorique pour f_1, et avec le résultat numérique fourni par la fonction `odeint` du module `scipy.integrate` pour f_2.

La fonction `odeint` prend en paramètre la fonction f, la valeur y_0 de y au point initial, et un tableau `abscisses` des valeurs de t en lesquels calculer une valeur approchée de y ; la première valeur de ce tableau correspond au point initial, en lequel y prend la valeur y_0. Le résultat renvoyé est un tableau (`array`) de la classe `numpy`, dont chaque entrée est elle-même un tableau contenant la valeur approchée de y à l'ascisse correspondante du tableau `abscisse`. Si on lance la fonction `odeint` sur une équation vectorielle, le tableau interne est contitué de la liste des coordonnées de la solution vectorielle y.

Pour créer le tableau des abscisses, on pourra utiliser la fonction `linspace` de `numpy`, prenant en argument deux réels a et b et un entier n supérieur ou égal à 2, et retournant un tableau de n valeurs régulièrement réparties entre a et b, la première valeur étant a, la dernière valeur étant b. Ainsi, pour obtenir une subdivision en n intervalles, il faut passer $n+1$ en paramètre.

Pour chacune des fonctions tests, on représentera les courbes calculées par les méthodes implémentées et les courbes théoriques ou numériques obtenues par `odeint` sur un même graphe. On soignera la présentation de ce graphe, en utilisant les fonctions du module `matplotlib.pyplot`, en particulier les fonctions `title`, `grid`, `legend`, `axhline`, `axvline`, `xlabel`, `ylabel`. Au sujet de ces fonctions, on consultera l'aide disponible.

On intéressera à l'influence du nombre de pas n, en faisant diverses représentations pour les valeurs de n suivantes : 2, 10, 50, 100.

1. Écrire une fonction `euler(f, a, y0, b, n)`, calculant numériquement sur $[a,b]$ la solution du problème $y' = f(t,y)$ vérifiant $y(a) =$`y0`, en utilisant une subdivision de l'intervalle en n, par la méthode d'Euler.
2. On note $h = \dfrac{b-a}{n}$, et pour tout $k \in [\![0,n]\!]$, $t_k = a + kh$.
   On rappelle qu'une méthode de Runge-Kutta, associée à une subdivision $t_k \leqslant x_1 \leqslant \cdots \leqslant x_\ell \leqslant t_{k+1}$, de poids $a_1, \ldots, a_{k+1}$ tels que $a_1 + \cdots + a_{k+1} = 1$ est décrite ainsi :
   - on calcule des pentes approchées intermédiaires $k_1, \ldots, k_\ell$, aux points $x_1, \ldots, x_\ell$, la pente k_{i+1} étant calculée par f au point x_{i+1}, en utilisant la valeur approchée de $y(x_{i+1})$ obtenue en approchant la courbe de y par la droite issue de (t_k, y_k) et de pente k_i (k_0 étant la pente initiale en t_k).
   - On calcule alors y_{k+1} en approchant la courbe par la droite issue de (t_k, y_k) et de pente p égale à la moyenne des k_i, avec les pondérations a_i.

   Écrire une fonction `RK2(f, a, y0, b, n)`, répondant au problème de la question 1, utilisant cette fois la méthode de Runge-Kutta d'ordre 2. Cette méthode est basée sur la subdivision donnée par l'unique réel $x_1 = t_k + \dfrac{h}{2}$. La seule pondération possible est $a_1 = 1$
3. Écrire une fonction `RK4(f, a, y0, b, n)`, répondant au problème de la question 1, en utilisant la méthode de Runge-Kutta d'ordre 4. Cette méthode est basée sur la subdivision donnée par $x_1 = t_k$, $x_2 = x_3 = t_k + \dfrac{h}{2}$ et $x_4 = t_{k+1}$. Les pondérations sont respectivement $\dfrac{1}{6}$, $\dfrac{2}{6}$, $\dfrac{2}{6}$ et $\dfrac{1}{6}$.

4. On définit l'erreur
$$e_n = \max_{k\in[\![0,n]\!]} |y_k - y(t_k)|.$$
Écrire une fonction `erreur` prenant en paramètre une méthode de calcul, un nombre de subdivisions n, et retournant l'erreur e_n dans le cas de la fonction f_1, avec les conditions et les bornes précisées au début de l'énoncé.
5. Écrire une fonction `rang_necessaire` prenant en paramètre une méthode de calcul, une marge d'erreur `eps` et retournant la plus petite valeur de n pour laquelle l'erreur e_n est inférieure à `eps`, dans le cas de la fonction f_1 (mêmes conditions initiales, mêmes bornes).

Exercice 2 – Équation du pendule

L'*équation du pendule* modélise les oscillations d'un pendule pesant dans un champ de pesanteur soumis à une force de frottement fluide proportionnelle à la vitesse : $x''(t) + \sin x(t) + \frac{1}{4}x'(t) = 0$.

Pour en faire une résolution numérique, il est nécessaire de préciser les conditions initiales : $x(0) = 0$, $x'(0) = v$ (on attribuera plus loin différentes valeurs numériques à la vitesse initiale v).

On utilisera la fonction `odeint` du module `scipy.integrate` pour la résolution des équations différentielles. Cette fonction peut être utilisée pour résoudre équations différentielles vectorielles $Y' = F(Y,t)$. Les données vectorielles doivent alors être rentrées sous forme d'un tableau de coordonnées.

On soignera les représentations graphiques, en particulier en annotant les axes, et en indiquant une légende.

1. Résoudre numériquement puis faire le tracé des solutions pour $v = 2$ et pour $v = 4$ sur l'intervalle $t \in [0, 30]$. Décrire qualitativement le mouvement du pendule dans chacun de ces deux cas.
2. Tracer ensuite le *diagramme des phases*, c'est-à-dire les courbes paramétrées $\begin{cases} x = x(t) \\ y = x'(t) \end{cases}$ pour chacune des deux conditions initiales données.
3. Écrire une fonction donnant une valeur approchée à 10^{-10} près de la vitesse initiale minimale v à donner au pendule pour qu'il fasse au moins k tours complets avant de se stabiliser ($k \in \mathbb{N}^*$)

Exercice 3 – Équation de Van der Pol

L'équation de Van der Pol régit le comportement de certains oscillateurs à un degré de liberté ; en particulier, cette équation peut servir à modéliser les battements du cœur :
$$x''(t) - \mu\left(1 - x(t)^2\right)x'(t) + x(t) = 0$$
L'étude de cette équation montre qu'il y a en général convergence vers un régime périodique indépendant des conditions initiales.

1. **Dépendance des conditions initiales**. On commence tout d'abord par fixer les valeurs $\mu = 1$ et $x(0) = 0$. Tracer sur un même graphe les différentes solutions correspondant aux conditions initiales $x'(0) = 0.001$, $x'(0) = 0.01$, $x'(0) = 0.1$ et $x'(0) = 1$ pour $t \in [0, 30]$, puis superposer sur une autre figure les différents diagrammes de phase.
2. **Dépendance du paramètre μ**. On fixe maintenant les valeurs $x(0) = 2$ et $x'(0) = 1$. Superposer les tracés obtenus pour différentes valeurs de μ prises entre 0 et 4.

   On peut observer que la période du régime permanent dépend de μ. On admettra que cette période est égale à la différence des abscisses de deux maximums consécutifs. Définir une fonction calculant la liste des maximums de x pour $t \in [0, 30]$ puis retournant la valeur moyenne de la période. Tracer alors le graphe représentant la valeur de la période en fonction de μ pour $\mu \in [0, 4]$.
3. **Oscillations forcées**. Lorsque cet oscillateur est excité par un terme harmonique de pulsation ω, l'équation différentielle devient :
$$x''(t) - \mu\big(1 - x(t)^2\big)x'(t) + x(t) = A\sin(2\pi\omega t)$$
   Observer le comportement de cet oscillateur pour $\mu = 8.53$, $A = 1.2$ et $\omega = 0.1$.