Université Lyon 1 Année 2020-2021
Master Mathématiques Générales 1ère année
Logiciels Scientifiques

TP3 – Option B

1 Étude d'EDO

Pour tout le TP, on pourra entrer les commandes

```
import numpy as np
import scipy.integrate as spi
import matplotlib.pyplot as plt
```

Exercice 1 - Schémas numériques

On considère l'EDO suivante

$$\begin{cases} y'(t) = ay(t) \\ y(0) = y_0 \end{cases}$$

dont la solution est $y(t) = y_0\, e^{at}$.

On se propose d'étudier les solutions numériques de cette équation. On considère une discrétisation uniforme de l'intervalle en temps $[0, T]$ et on note $t_i = ih$, $i \in [\![0, n]\!]$, avec $h = T/n$. Il s'agit de construire une approximation $\bar{y}_i \approx y(t_i)$ pour tout $i \in [\![0, n]\!]$.

Dans la suite, on considèrera les valeurs numériques $a = -1$, $y_0 = 1$, $T = 1$, $n = 10$.

1. **Méthode explicite à un pas -** L'approximaton $\bar{y}_i$ est définie comme suit :

$$\begin{cases} \bar{y}_0 = y_0 \\ \bar{y}_{i+1} = \bar{y}_i + h\, a\, \bar{y}_i, \quad i \in [\![0, n]\!]. \end{cases}$$

   Écrire une fonction `EulerEx(a,y_0,h,n)` donnant la liste des $\bar{y}_i$ correspondant à la méthode.

2. **Méthode implicite à un pas -** L'approximaton $\bar{y}_i$ est définie comme suit :

$$\begin{cases} \bar{y}_0 = y_0 \\ \bar{y}_{i+1} = \bar{y}_i + h\, a\, \bar{y}_{i+1}, \quad i \in [\![0, n]\!]. \end{cases}$$

   Écrire une fonction `EulerIm(a,y_0,h,n)` donnant la liste des $\bar{y}_i$ correspondant à la méthode.

3. **Méthode de Runge-Kutta d'ordre 2 -** L'approximaton $\bar{y}_i$ est définie comme suit :

$$\begin{cases} \bar{y}_0 = y_0 \\ k_1 = a\, \bar{y}_i \\ k_2 = a\,(\bar{y}_i + \dfrac{h}{2}\, k_1) \\ \bar{y}_{i+1} = \bar{y}_i + h\, k_2, \quad i \in [\![0, n]\!]. \end{cases}$$

   Écrire une fonction `RK2(a,y_0,h,n)` donnant la liste des $\bar{y}_i$ correspondant à la méthode

4. **Méthode de Runge-Kutta d'ordre 4 -** L'approximaton $\bar{y}_i$ est définie comme suit :

$$\begin{cases} \bar{y}_0 = y_0 \\ k_1 = a\, \bar{y}_i \\ k_2 = a\,(\bar{y}_i + \dfrac{h}{2}\, k_1) \\ k_3 = a\,(\bar{y}_i + \dfrac{h}{2}\, k_2) \\ k_4 = a\,(\bar{y}_i + h\, k_3) \\ \bar{y}_{i+1} = \bar{y}_i + \dfrac{h}{6}\,(k1 + 2k_2 + 2k_3 + k_4)\,, \\ \qquad\qquad\qquad\qquad i \in [\![0, n]\!]. \end{cases}$$

   Écrire une fonction `RK4(a,y_0,h,n)` donnant la liste des $\bar{y}_i$ correspondant à la méthode.

5. On considère $a = -1$, $y_0 = 1$, $T = 1$, $n = 10$. Tracer la solution exacte et les solutions approchées obtenues par les quatre méthodes sur une même figure en mettant un titre et une légende.

6. Calculer l'erreur commise pour chacune des méthodes en $t = T$.

Exercice 2 - Système proie-prédateur

Soit $l(t)$ le nombre de lapins et $c(t)$ le nombre de coyotes présents à un temps t donné. On considère les données suivantes :

- pour les lapins : taux de natalité=1/2, taux de mortalité=$c/50$ correspondant aux lapins dévorés par les coyotes
- pour les coyotes : taux de natalité=$l/10000$ proportionnel à la nourriture disponible, taux de mortalité=1/10).

Le système d'équations différentielles correspondant à l'évolution des deux populations pour $t > 0$ est alors le suivant :

$$\begin{cases} \dfrac{\mathrm{d}l}{\mathrm{d}t} = l\left(\dfrac{1}{2} - \dfrac{c}{50}\right) \\ \dfrac{\mathrm{d}c}{\mathrm{d}t} = c\left(\dfrac{l}{10000} - \dfrac{1}{10}\right) \end{cases}$$

1. Déterminer les points d'équilibre théoriques du système.
2. Écrire le problème sous la forme $\dfrac{\mathrm{d}x}{\mathrm{d}t} = f(x,t)$.
3. Écrire une fonction `f(x,t)` correspondant au problème.
4. Résoudre numériquement le système d'équations différentielles sur $[0, 50]$ pour différentes données initiales.
   ↪ *Utiliser la commande* `spi.odeint`.
5. Tracer l'évolution du système $(l(t), c(t))$ pour les différentes données initiales testées, ainsi que les points d'équilibre du système.
6. Le champ de vecteurs correspondant au système est le suivant :

$$(x,y) \mapsto \left[x\Big(\frac{1}{2} - \frac{y}{50}\Big), y\Big(\frac{x}{10000} - \frac{1}{10}\Big)\right]$$

   Définir deux vecteurs X et Y discrétisant respectivement $[0, 2600]$ et $[0, 50]$ avec $N = 25$ points, puis construire les tableaux U et V tels que $(U[i,j], V[i,j])$ contienne le champ de vecteur au point $(X[j], Y[i])$.
7. Tracer le portrait de phase du système sur la même figure, en règlant la longueur des flèches. et matérialiser les points d'équilibre par des croix rouges.
   ↪ *Utiliser la commande* `plt.quiver` *avec l'option* `angles='xy'`.

Exercice 3 - Oscillateur de Van der Pol *

On s'intéresse à la résolution numérique de l'équation différentielle de l'oscillateur de Van der Pol. Cette équation du second ordre s'écrit :

$$\ddot{x}(t) - \mu\left(1 - x(t)^2\right)\dot{x}(t) + x(t) = A\sin(\omega t)$$

où $\ddot{x}(t)$ est la dérivée seconde de x au temps t, μ est un paramètre physique, et $A\sin(\omega t)$ est un terme de forçage.

1. Écrire cette EDO du second ordre sous la forme d'un système d'EDO du premier ordre. On pourra poser $y = \dot{x}$ et écrire l'EDO vérifiée par (x, y).
2. Mettre ce système d'EDO du premier ordre sous la forme $\dot{Y} = f(t, Y)$ où $Y = (x, y)$. Écrire la fonction `f(t, Y)` du système. On prendra $\mu = 8.53$, $A = 1.2$ et $\omega = 2\pi/10$.
3. Écrire une fonction `Heun(f, Y_0, h, n)` résolvant une EDO avec la méthode de Heun qui s'écrit come suit :

$$\begin{cases} \bar{Y}_0 = Y_0 \\ k_1 = f(t_i, \bar{Y}_i) \\ k_2 = f(t_i + h, \bar{Y}_i + h\,k_1) \\ \bar{Y}_{i+1} = \bar{Y}_i + h\left(\dfrac{k_1}{2} + \dfrac{k_2}{2}\right) \quad i \in [\![0, n]\!]. \end{cases}$$

4. Tester la méthode pour $n \in \{1000, 5000, 10000\}$. On prendra $T = 75$. On pourra tracer, pour chacune de ces valeurs, la solution $x(t)$ en fonction du temps t et la solution dans le plan de phase $(x(t), y(t))$.

L'oscillateur de Van der Pol est un problème raide : la dérivée de la solution peut prendre des valeurs très grandes, i.e. la dérivée première varie très rapidement. Il faut donc prendre des pas de temps très petits pour capter ces variations rapides. Par ailleurs, le problème est numériquement mal posé, c'est à dire que, comme vous avez pu le remarquer, pour des pas de temps trop grands, la solution numérique explose. Ce serait le cas avec n'importe quel schéma numérique explicite.
Il existe des méthodes qui permettent d'adapter le pas de temps à chaque itération afin d'en prendre des petits dans les zones où la solution est raide, et des plus grand dans les zones ou la solution l'est moins. Celle que nous allons mettre en œuvre consiste à :

- prédire le prochain point $\bar{Y}_{i+1}^{Euler}$ avec une méthode d'Euler explicite (ordre 1) ;
- prédire le prochain point $\bar{Y}_{i+1}^{Heun}$ avec une méthode de Heun (ordre 2) ;
- calculer l'erreur entre ces deux méthodes $E = \|\bar{Y}_{i+1}^{Heun} - \bar{Y}_{i+1}^{Euler}\|$;
- si l'erreur est trop grande ($E > \epsilon_{\max}$), on diminue le pas de temps h et on recommence (on le divise par 2 par exemple). Si l'erreur est petite ($E < \epsilon_{\min}$), on augmente le pas de temps h en le multipliant par exemple par 2, et on recommence.

5. En remarquant que dans la méthode de Heun, le calcul de k_1 correspond à la méthode d'Euler explicite, écrire l'expression de l'erreur E en fonction de h, k_1 et k_2.
6. Écrire une fonction `AdaptHeun(f, Y_0, hinit, T, emin, emax)` qui résout l'EDO en adaptant le pas de temps et qui retourne le vecteur contenant tous les pas de temps utilisés ainsi que la solution $\bar{Y}$ à chaque pas de temps.
   ↪ *Question difficile !*
7. Résoudre le système d'EDO en prenant $\epsilon_{\min} = 10^{-4}$ et $\epsilon_{\max} = 10^{-2}$. Afficher le nombre de pas de temps effectués et tracer la solution.

---

2 Approximation numérique

Exercice 1 - Intégration numérique

L'intégrale I d'une fonction f entre a et b peut être approchée en utilisant une approche naïve (mais efficace) de l'intégrale. Les formules correspondant à quelques unes de ces méthodes sont les suivantes :

— Rectangles à gauche : $I \approx h \sum_{0 \leq i \leq n-1} f(x_i)$

— Rectangles à droite : $I \approx h \sum_{1 \leq i \leq n} f(x_i)$

— Trapèzes : $I \approx h\Big(\frac{f(a)+f(b)}{2} + \sum_{1 \leq i \leq n-1} f(x_i)\Big)$

où n est le nombre de points utilisés pour calculer l'intégrale, $h = (b-a)/n$ et $x_i = a+ih$ pour $i \in [\![0, n]\!]$.

1. Pour chacune des approches ci-dessus, écrire une fonction qui calcule l'intégrale d'une fonction f entre a et b.
2. Tester les fonctions avec, $a = 0$, $b = \pi/2$, $n = 100$, $h = (b-a)/n$ et $f(x) = \sin(x)$. Quelle est la méthode la plus efficace ?
3. Le package permettant de faire des intégrations approchées est `integrate`, dans la bibliothèque `scipy`. Entrer la commande `import scipy.integrate as integ` et tester la fonction `integ.quad`.

Exercice 2 - Interpolation polynomiale

1. Construire le polynôme $P = 2x^2 - 5x + 3$ à l'aide de la commande `np.poly1d`.
2. Quelles commandes permettent de récupérer
   — le degré d'un polynome P ?
   — son coefficient X^k ?
   — ses racines ?
   — la valeur de P en un point ?
3. Tracer le polynôme P sur l'intervalle $[-10, 10]$.
4. Le package permettant de faire de l'interpolation est `interpolate`, dans la bibliothèque `scipy`.
   (a) Construire le polynôme de Lagrange passant par les points $(0, 2)$, $(1, 4)$ et $(2, -1)$. Pour cela, entrer la commande `import scipy.interpolate as interp`, puis utiliser la fonction `interp.lagrange`.
   (b) Tracer le polynôme obtenu sur $[-1, 3]$, en ajoutant les points d'interpolation matérialisés par des croix rouges.

Exercice 3 - Phénomène de Runge

On considère une fonction f et son interpolé de Lagrange P_n d'ordre n aux points $(x_0, \cdots, x_n)$. Si f est C^{n+1} sur $[x_0, x_n]$, on rappelle que l'erreur d'interpolation au point $x \in [x_0, x_n]$ est donnée par

$$\exists \xi \in]x_0, x_n[, f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i).$$

Si les dérivées prennent des valeurs élevées, l'erreur d'interpolation peut donc devenir très importante. Le choix des nøeuds d'interpolation a également une grande influence.

Pour illustrer cela, on considère la fonction suivante sur $[-5, 5]$

$$f : x \mapsto \frac{1}{1+x^2}$$

1. Tracer sur un même graphique la fonction f et son polynôme d'interpolation de lagrange P_N avec $N = 2, 4, 10, 15$. Qu'observez-vous ?
   ↪ *Cela s'appelle le phénomène de Runge.*
2. On considère à présent les $N+1$ points d'interpolations suivants, dits points de Tchebychev

   $$\xi_i = -5 + 5\left(1 + \cos\left(\frac{2i+1}{2(N+1)}\pi\right)\right)$$

   On note R_N le polynôme d'interpolation de f sur les points (ξ_i). Tracer sur un même graphique la fonction f et son polynôme d'interpolation R_N avec $N = 2, 4, 10, 15$. Qu'observez-vous cette fois-ci ?
   ↪ *On peut montrer que ce choix de points d'interpolation est optimal en terme de minimisation de l'erreur d'interpolation.*

---

3 Résolution de $f(x) = 0$

Exercice 1 - Méthode de dichotomie

La méthode de dichotomie est basée sur le théorème des valeurs intermédiaires : une fonction $f : [a, b] \to \mathbb{R}$ continue telle que $f(a)$ et $f(b)$ sont de signes contraires admet au moins un zéro dans l'intervalle $]a, b[$. Pour un $\epsilon > 0$ fixé, elle permet de trouver un intervalle de taille ϵ contenant un zéro d'une fonction f continue.
L'algorithme est le suivant :

- Étant donné deux points a et b tels que $f(a)$ et $f(b)$ sont de signes opposés.
- Tant que $b - a > \epsilon$:
  - Poser $m = (a + b)/2$.
  - Si $f(a) * f(m) \leq 0$ alors poser $b = m$, sinon poser $a = m$.

On remarquera qu'il n'y a besoin que de connaître le signe de $f(x)$, pas sa valeur.

1. Écrire une fonction `dichotomie(f, a, b, epsilon)` qui implémente l'algorithme ci-dessus et retourne l'intervalle $[a, b]$ final de taille ϵ.
2. En utilisant `dichotomie`, donner la valeur de $\sqrt{2}$ à 10^{-10} près.
3. Quelle est la vitesse de convergence de cette méthode ?

Exercice 2 - Méthode de Newton 1D

La méthode de Newton utilise la dérivée de f pour obtenir une vitesse de convergence plus élevée que la méthode de dichotomie.
L'algorithme pour une fonction 1D réelle est le suivant :

- Étant donné un point x_0 "proche" du zéro de f recherché, une erreur *epsilon* > 0 et un entier N :
- Pour un point x_n donné, le point x_{n+1} est défini comme l'intersection de la tangente à f en x_n avec l'axe des abscisse.
- Si $|x_{n+1} - x_n| < \epsilon$ alors une approximation à ϵ près du zéro recherché est x_{n+1}, sinon on recommence au point précédent.
- Si le nombre d'itération est plus grand que N, l'algorithme s'arrête.

1. Écrire l'expression du point x_{n+1} en fonction de x_n, $f(x_n)$ et $f'(x_n)$.
2. Écrire une fonction `newton(f, df, x0, epsilon, N)` qui implémente l'algorithme de la méthode de Newton et retourne tous les itérés de la suite x_n.
3. Tester la méthode de Newton pour trouver une approximation de la valeur de $\sqrt{2}$ à 10^{-10} près. Combien d'itérations ont été effectuées ?
4. Estimer numériquement la vitesse de convergence de la méthode. Pour cela, tracer le log de l'erreur à l'itération $k+1$ en fonction du log de l'erreur à l'itération k et estimer la pente à l'aide de la commande `np.polyfit`.
5. Refaire l'étude de la convergence avec la fonction $f(x) = x^3 - 3x + 2$ et le zéro $x = 1$. Qu'observe t-on ? Pourquoi ?
6. Dans certain cas particulier, la méthode de Newton peut ne pas converger. Tester la méthode de Newton sur la fonction $f(x) = x^3 - 2x + 2$ avec $x = 0$ comme point initial. Que se passe t-il ?
7. On considère enfin la fonction $f(x) = x^{1/3}$. Que se passe t-il dans ce cas ? Tester pour différent point de départ.

Exercice 3 - Méthode de Newton multiD

Un autre avantage de la méthode de Newton est qu'elle fonctionne aussi pour des fonction de $\mathbb{R}^k$ dans $\mathbb{R}^k$ avec $k > 1$. Dans ce cas, le point x_n est mis à jour de la façon suivante :

$$x_{n+1} = x_n - D_f(x_n)^{-1} f(x_n)$$

où $D_f(x_n)$ est la différentielle de f au point x_n. Il s'agit donc de résoudre un système linéaire à chaque itération.

1. Adapter la méthode de Newton précédente pour qu'elle marche en dimension plus grande. On pourra utiliser la fonction `np.linalg.solve` pour résoudre le système linéaire.
2. Tester sur la fonction f de $\mathbb{R}^3$ dans $\mathbb{R}^3$ suivante :

$$f : (x, y, z) \mapsto \begin{bmatrix} 3x - \cos(yz) - 1.5 \\ 4x^2 - 625y^2 + 2z - 1 \\ 20z + e^{-xy} + 9 \end{bmatrix}$$

   et le point $x_0 = (1, 1, 1)$.
3. Comparer avec la solution donné par la fonction `solve` de `scipy.optimize`.
