TP4 POO - Python

simon.girel@univ-cotedazur.fr, gilles.scarella@univ-cotedazur.fr

1 Résolution d'équations différentielles ordinaires

1.1 EDO ordre 1

Dans le fichier *edo.py*, on veut résoudre l'équation différentielle suivante où u est une fonction dépendant du temps t, définie sur $[0, 3]$.

$$\begin{cases} 2\dfrac{du}{dt} - 3\,u &= (t+4)\,e^{3t/2} \\ u(0) &= -5 \end{cases} \tag{1}$$

- **Résoudre** en utilisant *scipy.integrate.odeint* l'EDO (1). L'intervalle $[0, 3]$ sera discrétisé par n points (n=21) et on appelle x le tableau scipy (ou numpy) contenant les points de discrétisation de $[0, 3]$.
  La solution exacte de l'EDO (1) est:

  $$u(t) = (\frac{1}{4}t^2 + 2\,t - 5)\ e^{3\,t/2}$$

  Définir une fonction python *uex* correspondant à cette solution exacte.

- **Afficher avec *matplotlib*** sur la même figure la solution exacte (en trait continu) et la solution numérique obtenue avec *scipy.integrate.odeint* (avec des points comme marqueurs).

1.2 EDO d'ordre 2

- Dans le même fichier python, **résoudre** l'EDO suivante avec *scipy.integrate.odeint*. L'intervalle $[0, 2]$ sera discrétisé par n points (n=21) et on appelle x le tableau scipy (ou numpy) contenant les points de discrétisation de $[0, 2]$.

$$\begin{cases} \dfrac{d^2y}{dt^2} - 4y &= 0 \\ y(0) &= 2; \quad \dfrac{dy}{dt}(0) = -1 \end{cases} \tag{2}$$

La solution exacte de cette EDO est:

$$y(t) = \frac{3}{4}e^{2t} + \frac{5}{4}e^{-2t}$$

Définir une fonction python *yex* correspondant à la solution exacte.

- **Afficher avec *matplotlib*** sur la même figure la solution exacte (en trait continu) et la solution numérique obtenue avec *scipy.integrate.odeint* (avec des points comme marqueurs).

2 Probabilités - TCL

Cet exercice est une application du Théorème Central Limite. Le code python associé sera écrit dans le fichier *tcl.py*.
Soient X_0, X_1, ..., X_i, .., X_{n-1} des variables aléatoires indépendantes et identiquement distribuées suivant la même loi. On notera μ l'espérance de X_i et σ l'écart-type (on suppose que ces deux valeurs sont finies et que l'écart type est non nul).
On définit la variable aléatoire S_n comme la somme de ces variables :

$$S_n = X_0 + X_1 + ... + X_{n-1}$$

Le TCL dit que, pour n assez grand, la loi normale $\mathcal{N}(n\mu, \sigma^2\, n)$ est une bonne approximation de la loi de S_n, qui a pour espérance $n\mu$ et écart-type $\sigma\sqrt{n}$.

Dans cet exercice, on se propose donc de comparer la densité de la loi normale à la densité empirique obtenues à partir de 2000 réalisation indépendantes de S_{40}. On va considérer ici que les variables aléatoires X_i suivent **une loi de Poisson $\mathcal{P}(\mu)$ de paramètre** $\mu = 8$ (d'espérance μ et d'écart-type $\sqrt{\mu}$). On considère donc $n = 40$ et on pose $M = 2000$, représentant le nombre de réalisations indépendantes.

- On considère un tableau numpy ou scipy T_X à 2 dimensions, formé de (n+1) lignes et M colonnes. Dans un premier temps, **définir T_X sur les n premières lignes** (et toutes ses M colonnes) à l'aide des fonctions statistiques de scipy. La k-ième colonne de T_X contient X_0 à X_{n-1} pour la k-ième répétition. Autrement dit la k-ième colonne de T_X doit contenir un tirage de n valeurs suivant la loi $\mathcal{P}(\mu)$.

- Sur la dernière ligne de T_X, calculez la somme S_n des n premières lignes de T_X, par colonne.

- Représenter la dernière ligne de T_X, correspondant à S_n, avec un histogramme normalisé (aire totale = 1). On utilisera un nombre d'intervalles (ou bins) égal à 50.

- Superposer à l'histogramme le graphe de la densité de la loi normale $\mathcal{N}(n\mu, \sigma^2\, n)$, en trait continu rouge. On utilisera une discrétisation en 100 points de l'intervalle allant de la plus petite à la plus grande valeur de la dernière ligne de T_X.

- Afficher la moyenne et l'écart-type de la dernière ligne de T_X.
- Même question en remplaçant la loi de Poisson par la loi binômiale $\mathcal{B}(30, 0.25)$.

3 Intégration numérique

3.1 Méthode des trapèzes

Dans un fichier *trapz.py*, **écrire une fonction python *my_ trapz*** qui approche l'intégrale de la fonction f sur $[a, b]$ par la méthode des trapèzes. On discrétise $[a, b]$ par n points régulièrement espacés définis comme suit

$$x_i = a + i\,\frac{(b-a)}{(n-1)}, \quad i = 0, ..., n-1$$

Le calcul de l'intégrale de f se décompose de la manière suivante

$$\int_a^b f(x)dx = \sum_{i=0}^{n-2} \int_{x_i}^{x_{i+1}} f(x)\,dx$$

On rappelle la formule de la méthode des trapèzes pour approcher l'intégrale sur l'intervalle $[x_i, x_{i+1}]$

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \simeq \frac{h}{2}\left(f(x_i) + f(x_{i+1})\right) \quad \text{où } h = (b-a)/(n-1)$$

On pourra utiliser une boucle for ou non dans cette fonction. La fonction possède 4 arguments (f, a, b et n) et renvoie l'approximation numérique de l'intégrale.

3.2 Exemple

On considère la fonction suivante sur l'intervalle [a,b]=[0,10]

$$f(x) = e^{-x^2}\sin(3x) - 4\,x\,\cos(x^2) + 2$$

Définir la variable python Iref obtenue en calculant avec *scipy.integrate.quad* la valeur de l'intégrale de f sur [0,10] (au besoin regarder l'aide de cette fonction). Cette valeur sera considérée comme la valeur exacte de l'intégrale dans la suite.

Utiliser votre fonction précédente *my_ trapz* et faites afficher la valeur approchée obtenue avec votre fonction. On prendra n=1001.

3.3 Affichage de l'erreur en échelle log

Définir une fonction python *plot_ erreur* prenant trois arguments: t_h, t_1 et t_2 Ces trois variables sont des tableaux numpy ou des listes de même taille.

Cette fonction affiche, avec la fonction *plt.loglog* de matplotlib, l'erreur $|t_1 - t_2|$ en fonction de t_h en échelle loglog. La figure contiendra le label 'h' pour l'axe des abscisses et aura le titre 'Erreur en fonction de h'.

De plus, la fonction doit afficher, avec *print*, la pente de la droite obtenue (on suppose qu'on obtient une droite) (on considère une échelle log, donc on calculera en fait le taux d'accroissement du graphe de $log(|t_1 - t_2|)$ en fonction de $\log(t_h)$.

On testera cette fonction avec les variables suivantes

```
import numpy as np
th = 0.01 * np.arange(1, 11)
t1 = np.arange(1, 11)
t2 = np.arange(1, 11) + th**2
```

3.4 Ordre de la méthode

Afin de déterminer l'ordre de la méthode, on définira trois tableaux pour les valeurs suivantes de n: n=51, 101, 1001, 5001, 10001, 50001, 100001

- v_1 contiendra les approximations numériques de l'intégrale pour chaque valeur de n (le pas de discrétisation change)
- v_2 est un tableau de même taille que v_1 contenant Iref, dans chaque élément
- v_h est le tableau des pas de discrétisation, chaque valeur vérifiant h=(b-a)/(n-1)

Utiliser la fonction *plot_erreur* appliquée à v_h, v_1 et v_2 afin de vérifier l'ordre de la méthode des trapèzes.

4 Résolution d'équations différentielles ordinaires [Facultatif]

4.1 EDO d'ordre 4

Résoudre avec *scipy.integrate.odeint* et représenter graphiquement la solution de l'EDO suivante sur l'intervalle $[0, 4]$

$$\begin{cases} \dfrac{d^4y}{dt^4} - 2\dfrac{d^2y}{dt^2} + \dfrac{dy}{dt} + y = 0 \\ y(0) = 1;\ \dfrac{dy}{dt}(0) = 0;\ \dfrac{d^2y}{dt^2}(0) = 1;\ \dfrac{d^3y}{dt^3}(0) = -1 \end{cases}$$