TD 2 : matplotlib

Quelques exercices de prise en main

1. Écrire un programme qui trace le graphe des fonctions $x \mapsto 1/x$ et cos sur $[-1, 1]$, l'une en pointillés rouges, l'autre solide et en bleu. Ajouter un titre, une légende, des labels pour chaque axe, et restreindre l'axe des ordonnées à $[-5, 5]$.
2. Écrire une fonction (de votre choix) qui prend en entrée un entier n et qui renvoit un vecteur de valeurs dépendant de n, puis générer un array de taille $4 \times n$ avec cette fonction, chaque ligne étant remplie grâce à la fonction.

   Écrire un programme qui crée quatre graphes en utilisant `plt.subplot`, où chaque graphe correspond à une ligne de l'array, sans oublier d'ajouter un titre à chaque subplot.
3. Générer un array de valeurs de la fonction $x \mapsto x^{-3/2}$, en ajoutant un bruit aléatoire (en utilisant `np.random.randn` par exemple). Tracer le graphe des valeurs obtenues en échelle logarithmique. Vaut-il mieux utiliser `plt.semilogx`, `plt.semilogy` ou `plt.loglog` ?

   Faire une régression linéaire avec `np.polyfit` sur les données obtenues, et tracer la droite correspondante sur le graphe.
4. Écrire un programme qui trace les lignes de niveau de la fonction $(x, y) \mapsto (x^2 + y^2)\cos(x)$, puis le champ de vecteur de son gradient.

Une équation de transport

On se propose d'étudier l'équation de transport suivante sur le tore $\mathbb{R}/2\pi\mathbb{Z}$

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{\partial u}{\partial x} = 0, & x \in [0, 2\pi],\ t > 0, \\ u(0, x) = \cos(x), & x \in [0, 2\pi]. \end{cases} \tag{E}$$

On discrétise le problème, et on applique la méthode d'Euler pour sa résolution approchée.

- Soit N un entier non nul, et $\Delta x = 2\pi/N$. Pour $i \in [\![0, N-1]\!]$, on pose $x_i = i\Delta x$.
- On choisit aussi un pas de temps Δt, et pour $n \in \mathbb{N}$, définit $t^n = n\Delta t$.
- On approche les dérivées partielles par des différences finies,

$$\frac{\partial u}{\partial x}(t, x) \simeq \frac{u(t, x + \Delta x) - u(t, x)}{\Delta x}, \quad \frac{\partial u}{\partial t}(t, x) \simeq \frac{u(t + \Delta t, x) - u(t, x)}{\Delta t}.$$

- On note u_i^n la solution approchée au temps t^n et au point x_i, et on suit un schéma d'Euler pour la résolution : pour tout $n \in \mathbb{N}$,

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{\Delta x}(u_i^n - u_{i-1}^n), \quad \forall i \in [\![0, N-1]\!]$$

  avec la convention $u_{-1}^n = u_{N-1}^n$.

1. Quelle est la solution de (E) ?
2. Écrire une fonction qui prend en entrée un réel T, et des entiers N et $M = T/\Delta t$, et qui renvoit le vecteur de la solution approchée donnée par le schéma d'Euler, $(u_i^M)_{i \in [\![0, N-1]\!]}$.
3. Tracer la solution avec `matplotlib.pyplot` pour plusieurs choix de paramètres (T, N, M). Ajouter un titre au plot, ainsi qu'une légende.
4. Écrire une fonction qui prend en entrée les paramètres T, N et M, et qui renvoit l'erreur, c'est-à-dire la norme infinie entre la solution approchée donnée par le schéma d'Euler et la solution exacte, au temps T.
5. Tracer l' erreur en fonction du pas de temps sur un graphe à échelle logarithmique avec `plt.loglog`

6. Effectuer une régression linéaire avec `np.polyfit` pour mettre en évidence la dépendance entre l'erreur et le pas de temps, puis ajouter la droite obtenue au graphe log-log.
7. Modifier le solveur basé sur le schéma d'Euler pour qu'il renvoit, en plus, le temps d'exécution de la fonction, puis refaire les questions 5. et 6. avec le temps d'exécution à la place de l'erreur. Vaut-il mieux tracer le temps d'exécution sur un graphe à échelle logarithmique, ou bien un graphe standard ?
8. (*) Faire une animation avec `plt.pause` ou la librairie `matplotlib.pyplot.animation` de la solution donnée par le schéma d'Euler au cours du temps.