TP6 (séance 7)

On présente deux modes d'utilisation des réseaux de neurones pour la résolution d'EDP extrêmement simples. On en reste au niveau de quelques principes.

1 Résolution d'une EDP elliptique par une méthode globale

Soit l'équation différentielle

$$-u''(z) = f(z), \quad 0 < z < 1,$$

avec les conditions aux deux bouts $u(0) = u(1) = 1$.

- On constitue un dataset en échantillonant des fonctions u

  $$u(z) = \sum_{i=1}^{p} a_i \sin(i\pi z), \qquad a_i = \frac{\text{random}(-10, 10)}{i\pi}$$

  de sorte que $u \in H^1(0,1)$ pour $p = \infty$. Ensuite $f(z) = -u''(z)$: en pratique on calcule

  $$f_j = -\frac{u_{j+1} - 2u_j + u_{j-1}}{\Delta x^2}$$

  Le dataset est $\mathcal{D} = \{x_r, y_r\}_{r=1,2,\ldots}$ est constitué de paires entrée-sortie de la forme $x = (f_j)_{1 \le j \le N}$ et $y = (u_j)_{1 \le j \le N}$ où N est le nombre de points de discrétisation, lequel est pris suffisamment grand.
  Charger et analyser `https://github.com/despresbr/NNNA/blob/main/lap_data.py`, puis exécuter.

- Charger et analyser `https://github.com/despresbr/NNNA/blob/main/lap_learn.py`, puis exécuter.

- Implémenter un test pour la fonction $u(x) = x(1-x)$ qui n'est pas constituée de sin.

- Pour les imaginatifs: trouver un réseau en ReLU efficace pour ce problème, au moins aussi efficace que celui avec la fonction d'activation linéaire utilisée dans le script.

- Généraliser au problème

$$-u''(z) + Cu(z)^3 = f(z), \quad 0 < z < 1,$$

avec les conditions aux deux bouts $u(0) = u(1) = 1$, avec une constante $C > 0$.

2 Résolution d'une équation de transport par une méthode locale

Soit l'équation d'advection

$$\partial_t u(t,z) + a\partial_x u(t,z) = 0, \quad 0 \le z \le 1,$$

avec des conditions aux bords périodiques

$$u(t,0) = u(t,1) \text{ pour tout temps } t \ge 0.$$

Nous nous concentrons sur des schémas dits en Volumes Finis de la forme

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + a\frac{u_{j+\frac{1}{2}}^{n+\frac{1}{2}} - u_{j-\frac{1}{2}}^{n+\frac{1}{2}}}{\Delta t} = 0.$$

Toute la question est la construction des **flux numériques** $u_{j+\frac{1}{2}}^{n+\frac{1}{2}}$ en fonction des valeurs u_k^n pour $k = j, j-1, j+2, j-2, j+2, \dots$ (au pas de temps n).

a) Rappeler pourquoi la solution exacte de l'équation d'advection est $u(t,x) = u_0(x - at)$.

b) Une idée naturelle est d'utiliser cette information en décidant que

$$u_{j+\frac{1}{2}}^{n+\frac{1}{2}} = u(j\Delta t - a\frac{1}{2}\Delta t, n\Delta t)$$

puis en construisant la valuer numérique de $u(j\Delta t - a\frac{1}{2}\delta t, n\Delta t)$ à partir des valeurs voisines u_k^n pour $k = j, j-1, j+2, j-2, \dots$. Autrement dit la construction du flux apparait comme un problème d'interpolation.

c) L'interpolation directe est construite grâce à un dataset où les entrées sont

$$x = (u((j-1)\Delta x), u(j+\Delta x), u((j+1)\Delta x))$$

(on peut aussi prendre $x = (u((j-2)\Delta x), u((j-1)\Delta x), u(j+\Delta x), u((j+1)\Delta x), u((j+2)\Delta x))$) et les sorties sont

$$y = u(j\Delta x - \frac{1}{2}a\Delta t) = u(j-\nu)\Delta x), \qquad \nu = a\frac{\Delta t}{\Delta x}.$$

Il ne reste plus qu'échantillonner les fonctions u.
Charger et analyser le script `https://github.com/despresbr/NNNA/blob/main/ML10_data.py` qui construit le dataset: plus précisément

$$x = (u((j-1)\Delta x) - u(j+\Delta x), 0, u((j+1)\Delta x) - u(j+\Delta x)).$$

d) Charger et analyser le script `https://github.com/despresbr/NNNA/blob/main/ML10_learn.py` pour effectuer le learning.

e) Enfin charger et analyser le script `https://github.com/despresbr/NNNA/blob/main/ML10_advection.py`. Tenter une étude de convergence en diminuant récursivement Δx d'un facteur 2 à chaque fois, et mesurer l'erreur numérique en fonction de Δx.
Nota Bene: les éventuels problèmes numériques rencontrés sont liés à la stabilité de ce schéma.