Université de Strasbourg
Cours Calcul Scientifique 2

TP 4 : Méthode des éléments finis

L'objectif de ce TP est de programmer la méthode des éléments finis (1D) pour résoudre l'équation elliptique avec conditions aux limites de Dirichlet:

$$\begin{aligned} -u''(x) + u(x) &= f(x), \qquad \text{sur }]0,1[\\ u(0) = u(1) &= 0. \end{aligned}$$

où f est la donnée et u est l'inconnue du problème. Commencez par importer les modules python :

```
import numpy as np
import scipy as sp
import scipy.sparse as spsp
import scipy.sparse.linalg as spsplin
```

Prendre soin de tester le code à chaque étape.

Partie 1. (Elements finis $\mathbb{P}_1$)

1. Définir une `class mesh` contenant le nombre d'éléments (sous-intervalles) du maillage `Nel`, le nombre de degré de liberté `Ndof` (degree of freedom), les points extrémaux `xmin`, `xmax`, le tableau contenant les noeuds du maillage `nodes`, le tableau contenant les tailles des mailles `h`.
2. Dans cette classe, définir une fonction `init_uniform(self)` et `init_random(self, Nel, xmin, xmax)` qui crée des maillages respectivement uniforme et aléatoire à partir de la donnée de `Nel`, `xmin` et `xmax`.
3. Définir une `class fem` qui contient le maillage `mesh`.
4. La matrice des éléments finis s'écrit :

$$A = (A_{i,j})_{i,j} = \left(\int_{[\text{xmin,xmax}]} \phi_i' \phi_j' + \phi_i \phi_j \right)_{i,j}$$

   où la (ϕ_i) est la base des éléments finis $\mathbb{P}_1$. Déterminer la taille de la matrice et calculer l'expression des coefficients en fonctions des tailles des éléments (h_j). Dans la classe `fem`, définir une fonction `matrixA_P1(self)` qui construit la matrice des éléments finis $\mathbb{P}_1$, en utilisant la **structure creuse diagonale**.
5. Toujours dans cette classe, définir la fonction `rhs_P1(self,f)` qui prend en entrée une fonction f et qui renvoie le tableau contenant une approximation du vecteur $b = (\int_{[\text{xmin,xmax}]} f\phi_i)_i$, où les intégrales sont calculées via la méthode des trapèzes par morceaux.
6. Pour tester la méthode, on cherche une solution "manufacturée" : déterminer $f(x)$ tel que $u(x) = \sin(\pi x)$ soit solution du problème.
7. Dans la classe `fem`, programmer une fonction `solve(self, f, plot=True)` qui prend en entrée une fonction f, qui calcule la solution approchée u par méthode des éléments finis et qui affiche la solution exacte et la solution approchée sur le même graphique si `plot` est `True`. Tester votre méthode sur la solution déterminée à la question précédente.

Partie 2. (Etude)

8. Inclure une fonction `norm_P1(self, u)` dans la classe mesh qui renvoie la norme L^2 et la semi-norme H^1 (discrète) de la fonction associée au vecteur u. Modifier ensuite la précédente `solve` pour qu'elle renvoie l'erreur en norme L^2 entre la solution approchée et la solution exacte.
9. Déterminer l'ordre de convergence numérique en traçant l'erreur en fonction du pas de maillage $h = \max h_j$ en échelle logarithmique (pour un nombre d'éléments parcourant la liste `[20,40,80,160,320,640]`). Qu'elle est l'ordre de convergence obtenu et celui qui était attendu ?
10. Afficher le conditionnement de la matrice (utiliser `np.linalg.cond`) pour le maillage uniforme puis pour le maillage aléatoire pour différentes valeurs de `Nel`. Commenter.

Partie 3 (Eléments finis $\mathbb{P}_2$).

11. Ajouter à la classe `mesh` l'attribut `deg` et le tableau `dof` (degree of freedom) de taille `deg*Nel+1` qui contient le points d'un maillage élément fini avec `deg+1` points répartis uniformément dans chaque maille ($x_{degj+k} = x_j + h_j k/deg$ pour $0 \leqslant j \leqslant Nel - 1$ et $0 \leqslant k \leqslant deg - 1$ et $x_{deg*Nel+1} =$ xmax). Modifier la définition du nombre de degré de liberté `Ndof`.
12. Définir une fonction `connect(self, el, k)` qui pour chaque numéro de l'élément et chaque numéro local du degré de liberté donne l'indice du degré de liberté associé.
13. Ajouter dans les fonctions `init_uniform` et `init_rand` la construction du tableau dof (on pourra faire une double boucle).
14. Nous voulons assembler la matrice suivante

$$A = (A_{i,j})_{i,j} = \left(\int_{[\mathrm{xmin},\mathrm{xmax}]} \phi_i' \phi_j' + \phi_i \phi_j \right)_{i,j}$$

où les $(\phi_j)_j$ sont les fonctions de base de la méthode d'élément fini $\mathbb{P}_2$. Nous rappelons que les fonctions de forme sur l'intervalle $[0,1]$ sont égales à :

$$\bar{\phi}_0(x) = (2x-1)(x-1), \quad \bar{\phi}_1(x) = 4x(1-x), \quad \bar{\phi}_2(x) = x(2x-1) \text{ sur } [0,1].$$

Les fonctions de base sont données par

$$\begin{aligned} \phi_{2j}(x) &= \bar{\phi}_2((x - x_{j-1})/h_{j-1})\, \mathbf{1}_{I_{j-1}}(x) + \bar{\phi}_0((x-x_j)/h_j)\, \mathbf{1}_{I_j}(x) \\ \phi_{2j+1}(x) &= \bar{\phi}_1((x-x_j)/h_j)\, \mathbf{1}_{I_j}(x) \end{aligned}$$

Vérifier que les matrices $M = (\int_{[0,1]} \bar{\phi}_k \bar{\phi}_{k'})_{0 \leqslant k,k' \leqslant deg}$ et $K = (\int_{[0,1]} \bar{\phi}_k' \bar{\phi}_{k'}')_{0 \leqslant k,k' \leqslant deg}$ sont données par

```
M = np.array([[2,1,-0.5], [1,8,1], [-0.5,1,2]]) / 15
K = np.array([[7,-8,1], [-8,16,-8], [1,-8,7]]) / 3
```

Sachant que l'on a :

$$\begin{aligned} A_{i,j} &= \sum_{\substack{\ell \in [1,Nel] \\ \sup \phi_i \cap \sup \phi_j \subset [x_\ell, x_{\ell+1}]}} \int_{[x_\ell, x_{\ell+1}]} \phi_i'(x)\phi_j'(x) + \phi_i(x)\phi_j(x) dx \\ &= \sum_{\substack{\ell \in [1,Nel] \\ \sup \phi_i \cap \sup \phi_j \subset [x_\ell, x_{\ell+1}]}} \int_{[0,1]} \left(\frac{\bar{\phi}_{n_{\ell,i}}'(y)}{h_\ell} \frac{\bar{\phi}_{n_{\ell,j}}'(y)}{h_\ell} + \bar{\phi}_{n_{\ell,i}}(y) \bar{\phi}_{n_{\ell,j}}(y) \right) h_\ell \, dy \end{aligned}$$

où $n_{\ell,i}$ et $n_{\ell,j}$ sont tels que $i = \text{connect}(\ell,n_{\ell,i})$ et $j = \text{connect}(\ell,n_{\ell,j})$, on construit la matrice en calculant toutes les contributions $\int_{[0,1]} \bar{\phi}'_{n_{\ell,i}}\bar{\phi}'_{n_{\ell,j}} + \bar{\phi}_{n_{\ell,i}}\bar{\phi}_{n_{\ell,j}}$ et en les ajoutant au bon coefficient de la matrice. Construire une fonction `matrixA(self)` qui assemble la matrice A en complétant les instructions suivantes:

```
for el in range( self.mesh.Nel):
    for ni in range(self.mesh.deg+1):
        i = self.mesh.connect(el, ni)
        for nj in range(self.mesh.deg+1):
            j = self.mesh.connect(el, nj)
            if 0 < i <= self.mesh.Ndof and 0 < j <= self.mesh.Ndof:
                A[i-1,j-1] += K[ni, nj] ...........
                A[i-1,j-1] += M[ni, nj] ...........
```

On pourra utiliser le format dok pour construire la matrice creuse. Afficher la matrice.

15. Ecrire une fonction `rhs(self, f)` qui prend en entrée une fonction f et qui renvoie le tableau contenant une approximation du vecteur $b = (\int_{[0,1]} f\phi_i)_i$, où les intégrales sont calculées via la méthode de Simpson par morceaux (parcourir les éléments de manière similaires à l'exemple ci-dessus).
16. Ecrire une fonction `norm(self,u)` dans la classe mesh qui renvoie la norme L^2 du vecteur u où les intégrales sont calculées à l'aide de la méthode de Simpson par morceaux.
17. Reprendre l'étude de convergence. Commenter.
18. [Facultatif] Reprendre les fonctions `matrixA`, `rhs` et `norm` pour qu'elles fonctionnnent aussi dans le cas des éléments finis $\mathbb{P}_1$.