INSA Département GM : TP Résolution de systèmes linéaires

A. Tonnoir (2018 - 2019)

Dans ce TP, on s'intéressera à la résolution numérique de l'équation suivante

$$-u''(x) + k(x)u(x) = f \quad \text{pour} \quad x \in (0,1)$$

Pour ce faire, on va chercher la solution approchée $u_i \simeq u(x_i)$ où $(x_i)_{i=0,n+1} = (ih)_{i=0,n+1}$, $h = 1.0/(n+1)$ est le *pas de discrétisation* et où les u_i vérifient le système d'équations :

$$-\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2} + k(x_i)u_i = f(x_i) \quad \forall i \in \{1, \cdots, n\} \tag{1}$$

Remarque : Deux semaines après la séance de TP, vous rendrez **vos codes complétés dans un dossier .zip et vos rapports de TP au format pdf**. Dans vos rapports, n'incluez pas vos codes !

Histoire de câble tendu

Dans cette première partie, nous allons chercher à résoudre l'équation dans le cas où $k(x) > 0$ et $f(x) = 0$. On modélise alors avec $u(x)$ la position d'un câble tendue entre deux points $x = 0$ et $x = 1$ et $k(x)$ représente la densité variable du câble. On supposera que $u_0 = u(0) = 1$ et $u_{n+1} = u(1) = 0$.

1. **(Théorique)**

   a) À l'aide d'un développement de Taylor de $u_{i\pm1} \simeq u(x_i \pm h)$ au voisinage de x_i, justifier simplement l'approximation (1) de l'équation différentielle.

   b) Écrire (1) sous la forme d'un système linéaire $A\underline{u} = \underline{b}$ où $\underline{u}^t = [u_1, \cdots, u_n]$ en précisant A et $\underline{b}$.

   c) Quelles sont les propriétés de la matrice A ? Quels algorithmes peut-on utiliser pour résoudre le système linéaire ?

2. **(Code)**

   a) Dans le fichier *data.py*, compléter la construction de la matrice A et du second membre $\underline{b}$.

   b) Dans le fichier *algo.py*, compléter les méthodes *cholesky* implémentant l'algorithme de Cholesky, *gradient* implémentant la méthode de descente de gradient et *CG* implémentant la méthode du gradient conjugué.

   c) Tester vos codes à l'aide du programme *main1.py*. Commenter vos résultats (précisions (comparer avec la solution exacte pour $k(x) = 1$), vitesse de calculs, ... soyez imaginatifs !).

Pour aller plus loin : Vous pouvez essayer d'optimiser vos codes de différentes manières. Par exemple, en utilisant des factorisations par bandes ou encore en exploitant le caractère creux des matrices pour optimiser les produits matrices vecteurs. Une autre piste de d'utiliser un preconditionneur pour les méthodes itératives. À vous de tester !

Du câble aux ondes, une histoire de signe

Dans cette deuxième partie, nous allons considérer $k(x) = -\omega^2$, ce qui nous ramène à l'équation d'Helmholtz. On va également changer les conditions aux bords et imposer des *conditions périodiques*. Pour ce faire, on considéra le système d'équation (1) pour $i \in \{\mathbf{0}, \cdots, n\}$ et on posera $u_0 = u_{n+1}$ et $u_{-1} = u_n$. Enfin, on supposera le terme source $f(x) \neq 0$.

1. **(Théorique)**

   a) Que modélise l'équation d'Helmholtz ?

   b) Avec les nouvelles *conditions aux limites* et $f(x) \neq 0$, reformuler le système (1) sous la forme d'un système linéaire $A\underline{u} = \underline{b}$ où $\underline{u}^t = [u_0, \cdots, u_n]$ en précisant A et $\underline{b}$ (attention, ce système sera de dimension $n+1$!).

   c) Quelles sont les propriétés de la matrice A ? Quels algorithmes peut-on utiliser pour résoudre le système linéaire ?

2. **(Code)**

   a) Dans le fichier *data.py*, compléter la construction de la matrice A et du second membre $\underline{b}$.

b) En supposant la matrice A inversible, implémenter dans *algo.py* une méthode de votre choix permettant de résoudre le système linéaire.

c) Tester vos codes à l'aide du programme *main2.py*. Commenter vos résultats.

Un jeu de détective !

Dans cette dernière partie, on va s'intéresser au *problème inverse* suivant : étant donné $k(x) = -\omega^2$ et la mesure $y_i = u(x_i)$ en m points $(x_i)_{i=1,m}$, retrouver la fonction source f qui a généré la solution $u(x)$ de l'équation d'Helmholtz. Pour aborder ce problème, on supposera que la solution de l'équation d'Helmholtz peut s'écrire / s'approcher sous la forme :

$$u(x) = \sum_{i=0}^{N} c_i \cos(2i\pi x) + \sum_{i=1}^{N} s_i \sin(2i\pi x) \tag{2}$$

où N est un paramètre fixé. On cherchera alors les coefficients c_i et s_i minimisant l'écart entre les données y_i et $u(x_i)$

1. **(Théorique)**

a) Justifier la recherche d'une solution $u(x)$ de la forme (2) (penser aux séries de Fourier).

b) Formuler le *problème de minimisation* ci-dessus sous la forme :

$$\text{Trouver } \underline{x} \in \mathbb{R}^{2N+1} \text{ minimisant } \quad \|A\underline{x} - \underline{y}\|_2^2$$

où on précisera $\underline{y} \in \mathbb{R}^m$ et A est une matrice $m \times (2N+1)$, le vecteur $\underline{x}$ étant $\underline{x}^t = [c_0, \cdots c_N, s_1, \cdots, s_N]$. Rappeler comment résoudre le problème.

c) Expliquer comment à partir de u on déduit le terme source f.

2. **(Code)**

a) Dans le fichier *data.py*, compléter la construction de la matrice A pour le *problème de minimisation*.

b) Dans le fichier *algo.py*, compléter l'algorithme d'orthonormalisation de Gram-Schmidt dans la fonction *qr* pour déduire la décomposition QR. Compléter ensuite la fonction *reconsF* pour calculer la reconstruction de la source f.

c) Tester vos codes à l'aide du programme *main3.py*. Commenter vos résultats.

Pour conclure le TP...

La dernière partie de ce TP porte sur la résolution d'un problème inverse. De manière générale, un *problème inverse* est un problème dans lequel on cherche à reconstituer les paramètres d'un modèle (ici la source f) connaissant une mesure (ici la déformation u). On retrouve ce type de situations dans de nombreuses applications telles que l'imagerie médicale, l'écholocalisation ou encore la restauration d'images. Mathématiquement, ces problèmes soulèvent de très nombreuses difficultés, à la fois théorique et numérique. Si vous êtes curieux d'en apprendre davantage, vous pouvez (par exemple) consulter ce cours...