Approximation par différences finies 2D

Loubna TALEB

02 Février 2023

Contents

1 Introduction

L'objectif du TP est de programmer l'équation différentielle suivante en espace-temps du problème suivant sur $\omega = [a_x, b_x] \times [a_y, b_y]$:

$$\begin{cases} \dfrac{\partial u(x,t)}{\partial t} - v\dfrac{\partial u(x,t)}{\partial x} = f(x,t) & \text{dans } \Omega \times [0,T] \\ u(x,t) = g(x,t) & \text{sur le bord } \partial\Omega \times (0,T) \\ u(x,0) = u_0(x) & \text{sur } \Omega \end{cases}$$

telles que f, g et u_0 sont les données du problème. Pour ce faire, nous avons discrétisé l'équation en temps en nous basant sur Euler implicite et Crank-Nicolson. Pour $n > 1$, nous avons :

Euler implicite :

$$\frac{1}{h_t}u^n(x) - \mu\Delta u^n(x) = f(x) + \frac{1}{h_t}u^{n-1}(x)$$

Crank-Nicolson :

$$\frac{1}{h_t}u^n(x) - \frac{\mu}{2}\Delta u^n(x) = f(x) + \frac{1}{h_t}u^{n-1}(x) + \frac{\mu}{2}\Delta u^{n-1}(x)$$

pour $h_t > 0$ et $t_n = nh_t$, où $u^n(x) = u(x, t_n)$.

Ensuite, nous avons discrétisé ces équations en espace en nous basant sur la méthode de Jacobi et Gauss-Seidel. En effet, les deux équations précédentes pour Euler implicite et Crank-Nicolson donnent l'équation suivante :

$$\alpha u^n(x) + \mu\Delta u^n(x) = f^n(x) \text{ dans } \Omega$$

$$u^n = g^n \text{ sur le bord } \partial\Omega$$

Donc, en appliquant l'approximation en 5 points, on trouve les deux méthodes itératives sans utiliser explicitement la matrice sous-jacente. Ainsi, pour :

Méthode de Jacobi :

$$u_{i,j}^{(k+1)} = \frac{1}{\lambda}\left[f_{i,j} + \frac{\mu}{h_x^2}(u_{i-1,j}^k + u_{i+1,j}^k) + \frac{\mu}{h_y^2}(u_{i,j-1}^k + u_{i,j+1}^k)\right]$$

où $\lambda = \alpha + \frac{2\mu}{h_x^2} + \frac{2\mu}{h_y^2}$.

Méthode de Gauss-Seidel :

$$u_{i,j}^{(k+1)} = \frac{1}{\lambda}\left[f_{i,j} + \frac{\mu}{h_x^2}(u_{i-1,j}^{k+1} + u_{i+1,j}^k) + \frac{\mu}{h_y^2}(u_{i,j-1}^{k+1} + u_{i,j+1}^k)\right]$$

2 Structure du Code

Nous avons créé trois fichiers : 'Functions.h' pour les définitions des fonctions et des variables globales, 'Functions.c' pour l'implémentation de ces fonctions, et 'main.c' pour calculer le temps d'exécution de chaque fonction et tester les fonctions implémentées. Nous avons également un 'makefile' pour compiler et exécuter le programme (vous pouvez simplement taper 'make' puis './my-program' pour tester et voir les résultats).

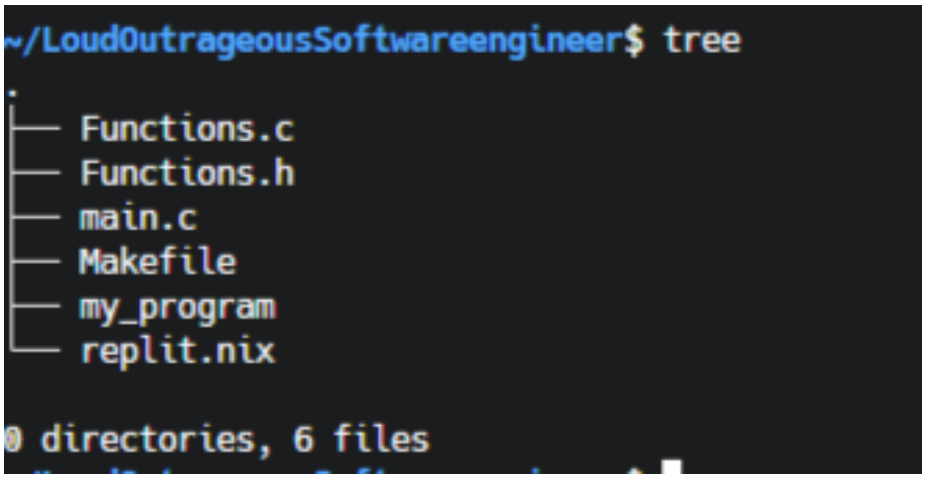

Figure 1: les Fichiers utilisée

2.1 Définitions des fonctions :

1. `index(int i, int j, int m)` : J'ai choisi de calculer $i \times (m+2) + j$ dans une fonction pour une meilleure lisibilité du calcul.
2. `square(double x)` : Pour calculer le carré d'un double.
3. `sol-exact(int i, int j, double hx, double hy)` : Pour calculer la solution exacte du premier problème, qui est : $u_e(x, y) = xy \sin(2\pi x) \cos(\pi y/2)$
4. `Jacobi(int m, int n)` : Pour implémenter la méthode itérative de Jacobi décrite ci-dessus.
5. `Gauss(int m, int n)` : Pour implémenter la méthode itérative de Gauss-Seidel décrite ci-dessus.
6. `Euler-implicite(int m, int n, int choix)` : Pour implémenter la méthode d'Euler implicite en se basant sur la discrétisation en espace de Jacobi. Il prend en paramètre un entier `choix` qui vaut 1 si l'utilisateur a choisi la méthode de Jacobi ou 2 s'il a choisi la méthode de Gauss-Seidel.
7. `Crank-Nicolson(int m, int n, int choix)` : Pour implémenter la méthode de Crank-Nicolson en se basant également sur la discrétisation en espace de Jacobi. Il prend en paramètre un entier `choix` pour choisir soit la méthode de Jacobi, soit la méthode de Gauss-Seidel.

2.2 Modèles testés:

Pour tester nos codes, nous avons testé deux modèles :

1. Celui-ci est écrit et testé dans la fonction du Jacobi et Gauss-Seidel :

$$\begin{cases} -\dfrac{\partial^2 u(x)}{\partial x^2} - \dfrac{\partial^2 u(x)}{\partial y^2} + u(x,t) = f(x) & \text{dans } \Omega = (0,1)^2 \\ u(x) = 0 & \text{sur le bord } \partial\Omega \end{cases}$$

sachant que $f(x) = \frac{17\pi}{4} 2xy \sin(2\pi x) \cos(\frac{\pi y}{2}) + \pi x \sin(2\pi x) \sin(\frac{\pi y}{2}) - 4\pi y \cos(2\pi x) \cos(\frac{\pi y}{2}) + xy \sin(2\pi x) \cos(\frac{\pi y}{2})$

2. Celui-ci est écrit et testé dans la fonction Euler-implicite et Crank-Nicolson :

$$\begin{cases} \dfrac{\partial u(x,t)}{\partial t} - \Delta u(x,t) = (a^2+b^2) w_0(x,t) & \text{dans } (0,1)^2 \times [0,T] \\ u(x,t) = w_0(x,y) & \text{sur le bord } \partial\Omega \times (0,T) \\ u(x,0) = w_0(x) & \text{sur } (0,1)^2 \end{cases}$$

sachant que $w_0(x,y) = \sin(ax)\cos(by)$ et $a = b = \pi$.

3 Résultats :

3.1 Jacobi:

On a testé la méthode de Jacobi pour plusieurs valeurs de n et m (mais puisque nous sommes dans une grille carrée donc $n = m$). Comme nous avons vu en cours, on prend $h = \frac{1}{2^n} = \frac{1}{m+1}$ donc nous avons testé : $m = 3, 7, 15, 31, 63, 127$ et nous avons bien trouvé que le nombre d'itérations se redouble aussi si on double h, et nous avons aussi trouvé que quand $h \to 0$, err $\to 0$, comme on le montre cette exécution.

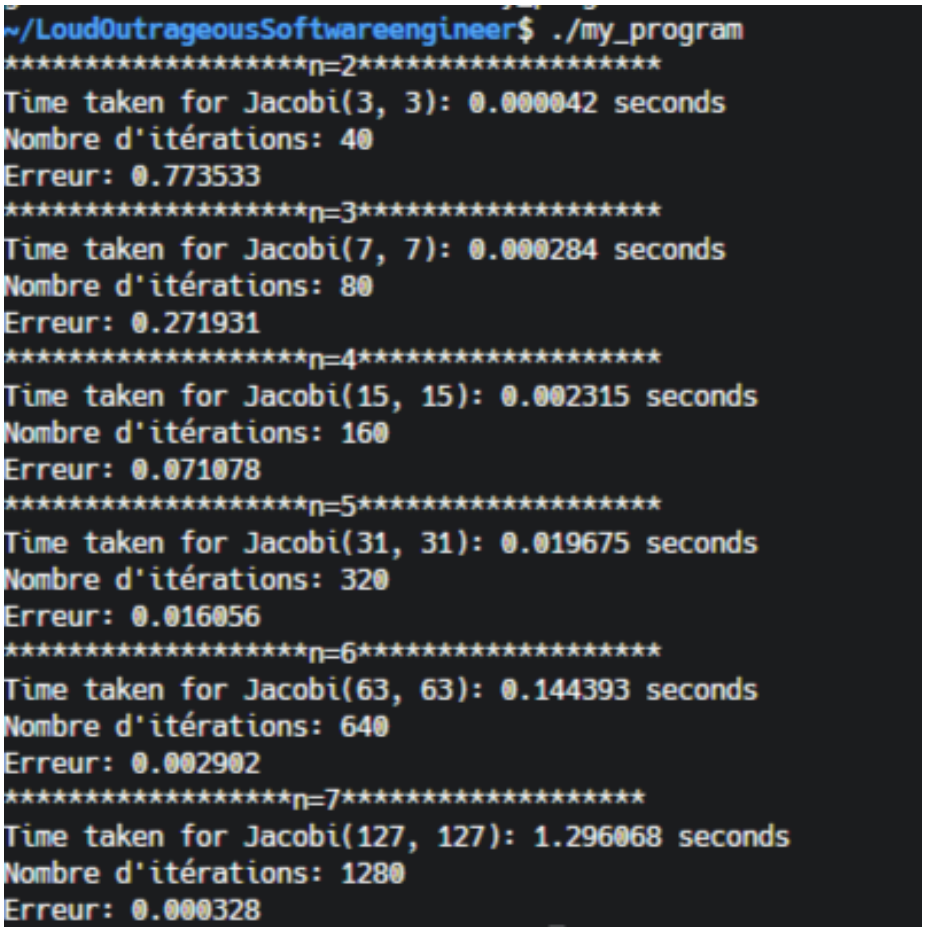

Figure 2: Exécution du Jacobi

3.2 Gauss-Seidel:

De même, nous avons testé la méthode itérative de Gauss-Seidel pour plusieurs valeurs de n, et nous avons trouvé aussi que le nombre d'itérations se redouble si h se redouble. De plus, l'erreur relative entre la solution exacte et la solution provenant de la méthode itérative tend vers zéro quand h tend vers zéro. De plus, nous avons trouvé que le temps d'exécution augmente si on augmente la taille de la grille. Voici les résultats après l'exécution :

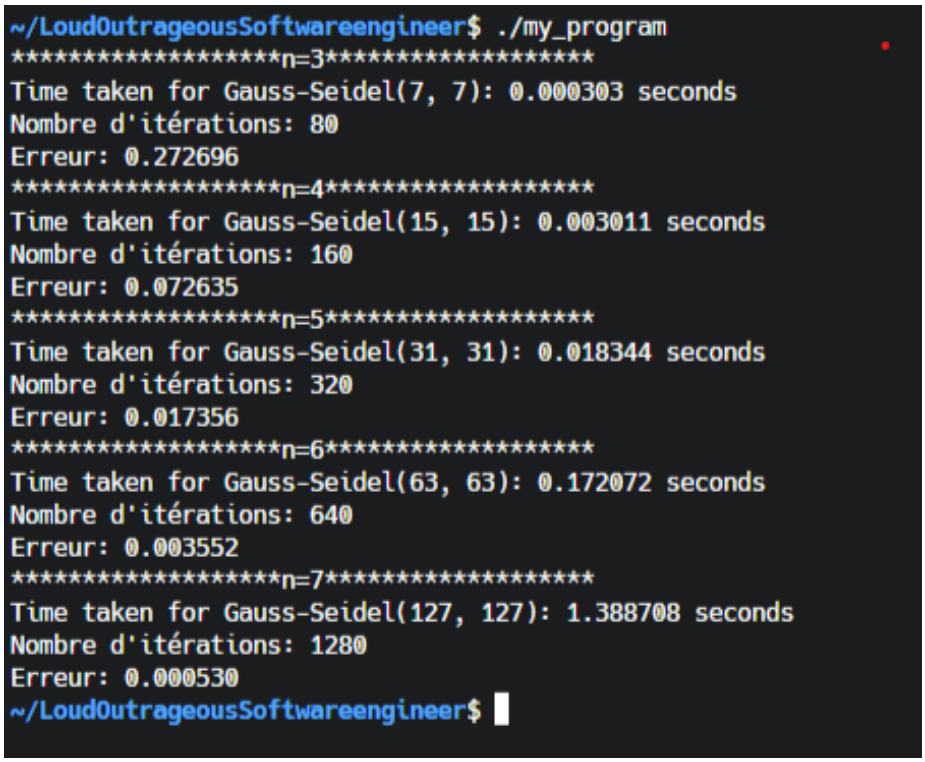

Figure 3: Exécution du Gauss Seidel

3.3 Euler-implicite:

En utilisant le schéma d'Euler implicite décrit ci-dessus, nous avons implémenté cette méthode en donnant le choix à l'utilisateur entre la méthode de discrétisation en espace de Jacobi ou bien de Gauss-Seidel. Pour ce faire, nous avons créé deux conditions sur le choix de l'utilisateur, dont le contenu est la boucle d'itération en espace de chacune des deux méthodes itératives, le tout dans la boucle d'itération sur le temps pour la méthode implicite.

Ensuite, nous avons testé la méthode pour plusieurs valeurs de n et m (sachant que $n = m$), et nous avons également calculé le temps d'exécution

pour la méthode.

Voici les résultats de l'exécution:

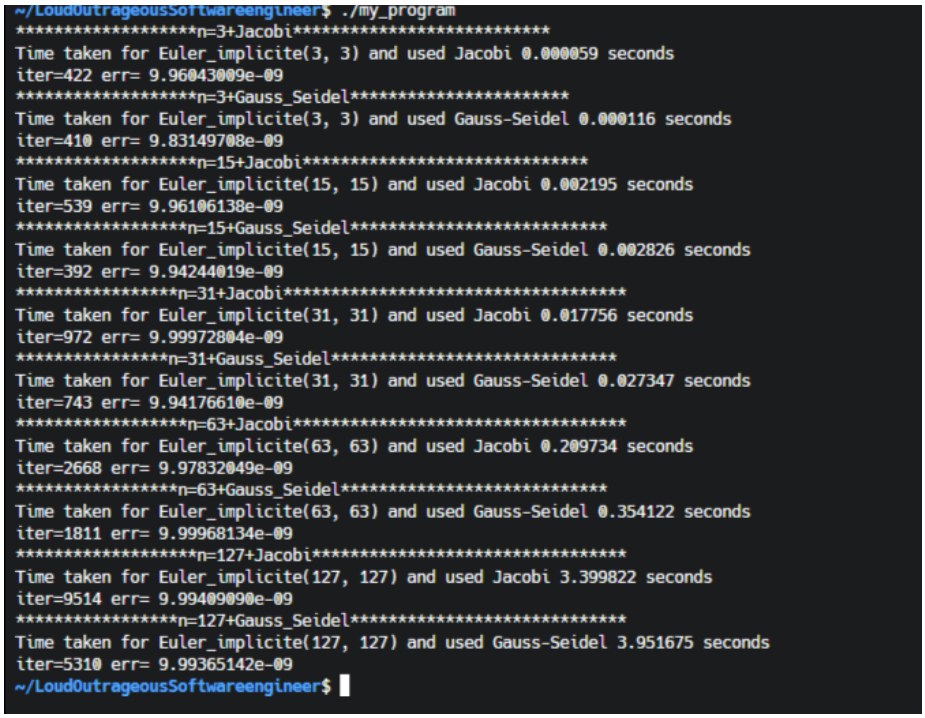

Figure 4: Exécution du Euler implicite

3.4 Crank-Nicolson:

Par le même principe, nous avons implémenté la méthode de Crank-Nicolson en donnant le choix aux utilisateurs entre la méthode de Jacobi et Gauss-Seidel, en se basant sur le principe de la méthode de Crank-Nicolson qui combine les deux méthodes d'Euler implicite et explicite (en divisant les deux termes par 2). C'est pourquoi j'ai multiplié le deuxième terme par 0.5.

Ensuite, nous avons testé la méthode pour plusieurs valeurs de n et m (sachant que $n = m$), et nous avons également calculé le temps d'exécution pour la méthode.

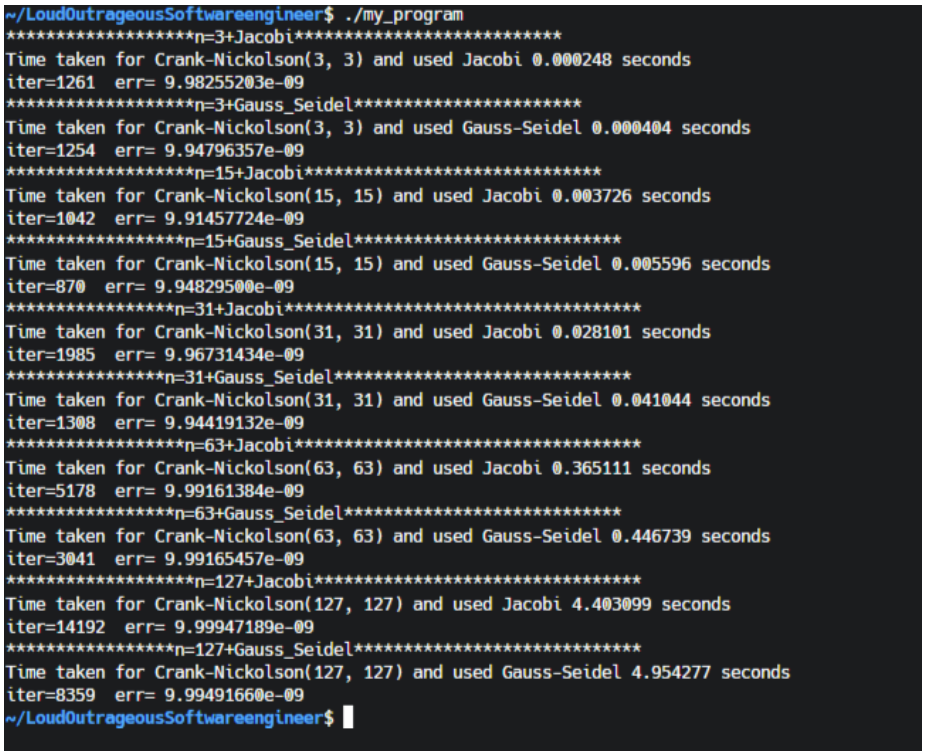

Figure 5: Exécution du Crack Nicolson

4 Conclusion:

En définitive, les 4 fonctions de discrétisation en espace-temps fonctionnent bien. Toutefois, un problème de durée d'exécution est survenu lorsque j'ai augmenté la taille de la grille, en raison de l'utilisation de la plateforme en ligne Replit. Replit offre un environnement de développement intégré (IDE) pour programmer dans différents langages de programmation, mais sa RAM et son CPU sont limités. C'est pourquoi j'ai testé jusqu'à 127 (en doublant la valeur de "iter" lorsque "h" est doublé). De plus, j'ai vidé la mémoire à la fin de chaque fonction pour éviter les problèmes de fuite de mémoire. Comme nous n'utilisons pas les tableaux de solution pour tracer "u" au cours du temps et en espace, cela n'a pas affecté le résultat final.