UVSQ

université PARIS-SACLAY

Rapport TP Poisson

Résolution du problème de Poisson avec différentes méthodes

Rédigé par :

Estelle Oliveira

Date : Décembre 2024

Le but de ce TP est de résoudre l'équation de la chaleur en 1D stationnaire. Nous appliquerons un ensemble d'algorithmes pour résoudre ce système linéaire obtenu par discrétisation par la méthode des différences finies de l'équation de la chaleur 1D stationnaire.

1 Introduction

Nous souhaitons résoudre l'équation de la chaleur dans un milieu immobile linéaire homogène isotrope avec un terme source. On pose alors le problème comme suit :

$$\begin{cases} -k\frac{\partial^2 T}{\partial x^2} = g, \quad x \in]0,1[\\ T(0) = T_0 \\ T(1) = T_1 \end{cases}$$

où g est un terme source,

$$k > 0$$

le coefficient de conductivité thermique, et

$$T0 < T1$$

les températures au bord du domaine considéré.

Dans la suite, on considérera qu'il n'y a pas de source de chaleur, la solution analytique de cette équation est donc :

$$T(x) = T0 + x(T1 - T0)$$

La résolution de l'équation de poisson se fera à l'aide d'un stockage de matrice en matrice bande. On ne garde que les éléments qui nous intéressent, éliminant dans notre cas les zéros. Dans certains cas une ligne de zéros supplémentaire peut être nécessaire pour effectuer les calculs. Cependant, le stockage en matrice bande permet un gain significatif, rendant le stockage et le traitement de grandes matrices possible et efficace.

2 Réponses aux questions

Pour résoudre efficacement l'équation de poisson, nous utiliserons les bibliothèques BLAS et LAPACK. Ces bibliothèques nécessitent une gestion spécifique des matrices, détaillé ci-après.

2.1 Question 1

En C comment doit on déclarer et allouer une matrice pour utiliser BLAS et LAPACK ? Pour utiliser BLAS et LAPACK, les matrices doivent respecter le stockage priorité colonne. Comme nous avons une matrice tridiagonale, celle-ci sera stockée dans un tableau compact, pour limiter le stockage. Ce sera alors une matrice dite matrice bande. L'allocation peut se faire de la manière suivante : "'c double* AB = (double*)malloc(M * N * sizeof(double)); "' avec M le nombre de lignes et N le nombre de colonnes.

2.2 Question 2

Quelle est la signification de la constante LAPACK-COL-MAJOR ? La constante LAPACK-COL-MAJOR fait référence au stockage en mémoire, en priorité colonne.

2.3 Question 3

A quoi correspond la dimension principale généralement notée ld ? Lorsqu'une matrice est stockée en colonne-major, la dimension principale notée ld correspond à la distance mémoire entre deux éléments consécutifs d'une colonne. Dans le cas d'une matrice n x m, cela correspond généralement à m.

2.4 Question 4

Que fait la fonction dgbmv ? Quelle méthode implémente-t-elle ? La fonction dgbmv dans BLAS est utilisée dans le cadre d'une multiplication matrice-vecteur d'une matrice bande. La fonction calcule alors :

$$y = \alpha Ax + \beta y$$

Elle est de complexité globale : O(n(kl+ku)).

2.5 Question 5

Que fait la fonction dgbtrf ? Quelle méthode implémente-t-elle ? La fonction dgbtrf de LAPACK est utilisée pour effectuer une factorisation partielle LU d'une matrice bande. La focntion calcule alors :

$$A = PLU$$

Elle est de complexité globale : O(n(kl+ku)2).

2.6 Question 6

Que fait la fonction dgbtrs ? Quelle méthode implémente-t-elle ? La fonction dgbtrs de LAPACK résout le système :

$$AX = B$$

en utilisant la factorisation générée par dgbtrf. Elle est de complexité globale : O(n(kl+ku)).

2.7 Question 7

Que fait la fonction **dgbsv** ? Quelle méthode implémente-t-elle ? La fonction dgbsv revient à faire la fonction **dgbtrf** suivie de la fonction **dgbtrs**. Elle effectue donc la factorisation LU suivie de la résolution du système en un seul appel de fonction.

2.8 Question 8

Comment calculer la norme du résidu relatif avec des appels BLAS ? On rappelle la formule de la norme du résidu :

$$\text{résidu relatif} = \frac{\|AX - B\|}{\|B\|}$$

On appelera alors la fonction dgemv pour calculer AX, puis on fait tout simplement la différence avec B. Ensuite, on pourra calculer la norme de AX - B, ainsi que celle de B, avec snrm2/scnrm2 ou dnrm2/dznrm2. Le résidu se calule ensuite en faisant la division entre les deux normes obtenues.

2.9 Mesures de performances

Les mesures de performances permettent de déterminer le temps écoulé pour résoudre avec dgbsv à environ 0.000037 secondes. Pour dgbtrf + dgbtrs, c'est résolu en 0.000047 secondes. L'erreur relative est la même entre les deux méthodes. On peut déterminer le rapport de temps entre les 2 methodes :

$$\text{rapport} = \frac{0.000047}{0.000037}$$

On a donc un rapport d'environ 1.27 entre les deux méthodes.

3 Fonctions pour libpoisson

1. set_GB_operator_colMajor_poisson1D permet d'initialiser la matrice bande de poisson qui nous permettera par la suite de réaliser tous nos calculs. Il s'agira de la matrice AB. Cette matrice aura la forme suivante :

$$M_{\text{bande}} = \begin{bmatrix} 0 & 2 & -1 \\ -1 & 2 & -1 \\ & \cdots & \\ -1 & 2 & 0 \end{bmatrix}$$

On rajoutera une colonne supplémentaire de zéros lorsque kv = 1. kv est un paramètre de largeur de bande, qui nous permettera ici de déterminer le nombre de colonnes nécessaires de zéros dont on aura besoin pour réaliser tous nos calculs.

2. set_GB_operator_colMajor_poisson1D_Id permet d'initialiser la matrice Identité format matrice bande de la même manière que la matrice de poisson.

3. set_dense_RHS_DBC_1D permet de poser les conditions de bord pour le vecteur RHS (right hand side). Les termes de bords sont initialisés et le vecteur est rempli avec des valeurs nulles.

4. set_analytical_solution_DBC_1D permet de calculer la solution dite "analytique" qui est la solution exacte, de référence du problème de Poisson. Elle sera utile pour comparer les différentes solutions calculées grâce aux différentes méthodes et les comparer à la solution exacte. Cela permettera d'avoir une vérification pour les différentes méthodes proposées dans ce TP.

5. set_grid_points_1D permet de déterminer la position de tous les points x[i] en fonction du pas h choisi.

6. relative_forward_error nous permet de calculer l'erreur relative avant, on utilise la formule :

$$\text{résidu relatif} = \frac{\|AX - B\|}{\|B\|}$$

on pose alors dans cette fonction, AX = x et B = y.

4 Fonctions pour Richardson

eigmax et eigmin permettent de déterminer les valeurs propres maximales et minimales d'une matrice associée au problème de Poisson 1D.

1. eigmax_poisson1D permet de calculer la valeur propre maximale à l'aide de la formule suivante :

$$4\left(\sin\left(\frac{\text{la}\cdot\pi\cdot h}{2}\right)\right)^2$$

2. eigmin_poisson1D permet de calculer la valeur propre minimale à l'aide de la formule suivante :

$$4\left(\sin\left(\frac{\pi\cdot h}{2}\right)\right)^2$$

avec h le pas de discrétisation, la le nombre de colonne de la matrice. Dans le contexte des matrices de Poisson, on obtient les valeurs propres en posant pour la valeur propre minimale, la = 1; puisque cela

nous renvoie à la première valeur de notre matrice.

3. eig_poisson1D permet de déterminer toutes les valeurs propres de la matrice.

4. richardson_alpha_opt permet de déterminer le alpha optimal pour la méthode richardson en utilisant les fonctions eigmin_poisson1D et eigmax_poisson1D. Dans la méthode de Richardson, pour résoudre un système linéaire, l'itération de mise à jour est donnée par :

$$x(k+1) = x(k) + alpha(b - Ax(k))$$

où alpha est le pas de Richardson. Le choix de alpha influence la rapidité de convergence de la méthode. A chaque itération, l'erreur évolue selon la relation :

$$e(k+1) = (I - alphaA)e(k)$$

Pour que la méthode converge rapidement, il faut minimiser la norme de $(I - alphaA)$, ce qui revient à choisir un alpha qui minimise l'opérateur spectral de $(I - alphaA)$. L'alpha optimal est celui qui va permettre de minimiser le nombre d'itérations nécessaires pour obtenir une solution avec une erreur inférieure à ce que l'on voulait. Dans le cas d'une matrice symétrique définie positive, on détermine alpha optimal grâce aux valeurs propres maximales et minimales de la matrice comme suit :

$$\frac{2.0}{eigmax\_poisson1D + eigmin\_poisson1D}$$

5. richardson_alpha permet de résoudre le problème à l'aide du alpha optimal déterminé plus tôt.

6. extract_MB_jacobi_tridiag permet d'extraire la matrice MB de Jacobi agissant ici comme matrice de préconditionnement qui permet une convergence plus rapide de la méthode de Jacobi. Le préconditionneur de Jacobi utilise uniquement la diagonale principale de AB, ainsi dans notre problème, MB sera une matrice bande de sorte à ce que l'on ait :

$$M_{\text{bande}} = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 2 & 0 \\ 0 & 2 & 0 \\ & \cdots & \\ 0 & 2 & 0 \end{bmatrix}$$

7. extract_MB_gauss_seidel_tridiag permet d'extraire la matrice MB de Gauss-Seidel agissant comme matrice de préconditionnement qui permet une convergence plus rapide de la méthode de Gauss-Seidel. Le préconditionneur de Gauss-Seidel utilise la diagonale principale de AB mais aussi la diagonale inférieure, ainsi dans notre problème, MB sera une matrice bande de sorte à ce que l'on ait :

$$M_{\text{bande}} = \begin{bmatrix} -1 & 2 & 0 \\ -1 & 2 & 0 \\ -1 & 2 & 0 \\ & \cdots & \\ -1 & 2 & 0 \end{bmatrix}$$

8. richardson_MB permet de résoudre le problème à l'aide des matrices de préconditionnement MB déterminées plus tôt pour les méthodes de Jacobi et Gauss-Seidel.

5 Stockage CSC et CSR

Lors de la résolution de systèmes linéaires, il existe plusieurs formats de stockages efficaces des matrices. Ces formats comme CSC (Compressed Sparse Column) et CSR (Compressed Sparse Row), sont utilisés pour mieux gérer la mémoire des matrices creuses.

5.1 CSC

Dans le stockage CSC, Compressed Sparse Column, une matrice est représentée en trois tableaux :

- Valeurs des éléments non nul
- Indice de lignes (ligne des éléments non nuls dans chaque colonne)
- Début de chaque colonne (commencement de chaque colonne dans le tableau de valeur)

5.2 CSR

Dans le stockage CSR, Compressed Sparse Row, une matrice est représentée en trois tableaux :

- Valeurs des éléments non nul
- Indice de colonnes (colonne des éléments non nuls dans chaque ligne)
- Début de chaque ligne (commencement de chaque ligne dans le tableau de valeur)

5.3 Richardson en stockage CSC et CSR

5.3.1 Initialisation

- Initialiser le vecteur solution x_0
- Calculer le résidu initial :

$$RHS = b - A \cdot x_0$$

Pour calculer le produit $A \cdot x_0$, il est nécessaire de parcourir chaque colonne/ligne de A dans le format CSC/CSR, en accédant aux éléments non nuls et en multipliant ces éléments par les valeurs correspondantes de x_0.

5.3.2 Boucle itérative

Tant que le critère de convergence n'est pas atteint, on effectue les étapes suivantes pour chaque itération k, tant que le nombre d'itérations maximum n'est pas dépassé :

- Calculer le produit matrice-vecteur $A \cdot x_k$ en utilisant le stockage CSC/CSR :

$$A \cdot x_k = \sum_{i=1}^{n} A_{ik} x_k$$

où A_{ik} est un élément non nul de la matrice, et x_k est l'élément correspondant du vecteur solution.

- Mettre à jour la solution :

$$x_{k+1} = x_k + \alpha \cdot (b - A \cdot x_k)$$

- Calculer le résidu :

$$RHS = b - A \cdot x_{k+1}$$

- Vérifier la convergence en calculant la norme du résidu :

$$\|RHS\| < \epsilon$$

où ϵ est un seuil de tolérance.

L'utilisation du format **CSC/CSR** pour le stockage de matrices dans la méthode de Richardson permet d'optimiser la mémoire et d'améliorer l'efficacité des calculs. C'est une très bonne approche pour résoudre des systèmes linéaires avec des matrices creuses de grande taille.

6 Convergence et performance

6.1 Richardson

La méthode de Jacobi nous permet d'avoir une convergence assez rapide, le résidu commence à 1.0; celui-ci tombe à 10e-3 en un peu plus de 120 itérations. Cependant, on peut observer sur le graphique de comparaison entre la solution calculée et la solution exacte une légère divergence.
Erreur relative calculée : 5.093853e-03.

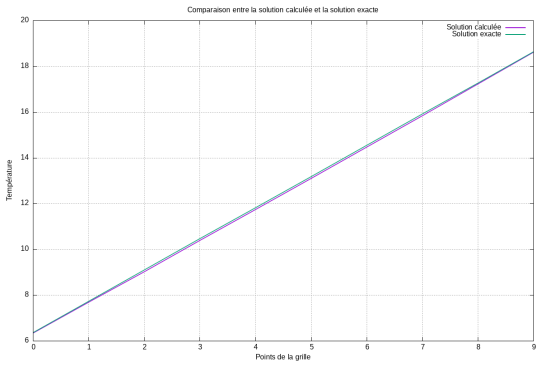

Figure 1: Comparaison de la solution exacte avec la solution calculée

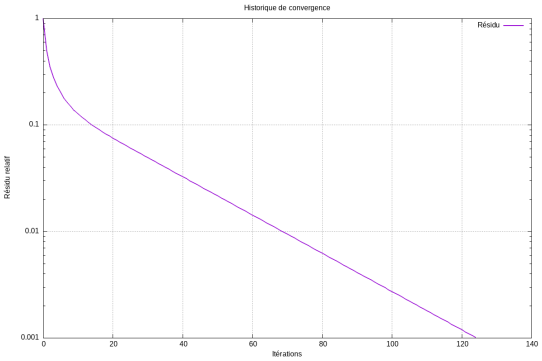

Figure 2: Convergence Richardson

6.2 Jacobi

La méthode Jacobi permet d'obtenir une solution plus proche de la solution exacte, pour autant on aura une convergence plus lente, le résidu commence à 10; celui-ci tombe à 10e-3 en un peu moins de 200 itérations.
Erreur relative calculée : 2.489418e-04.

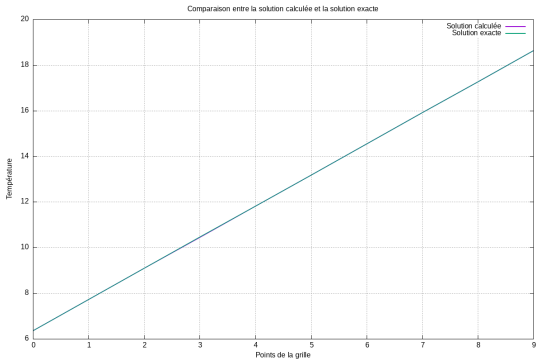

Figure 3: Comparaison de la solution exacte avec la solution calculée

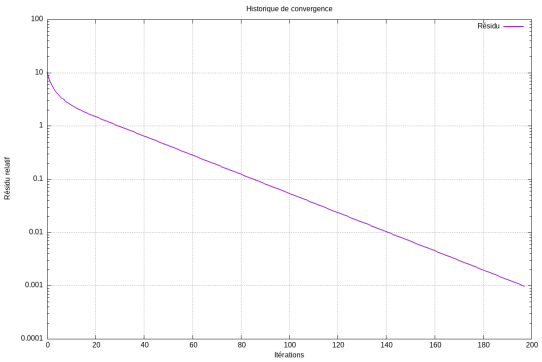

Figure 4: Convergence Richardson

6.3 Gauss-Seidel

La méthode Gauss-Seidel, comme Jacobi, permet d'obtenir une solution plus proche de la solution exacte, pour autant on aura une convergence plus lente, le résidu commence à 10; celui-ci tombe à 10e-3 en un peu moins de 200 itérations.
Erreur relative calculée : 2.489418e-04.

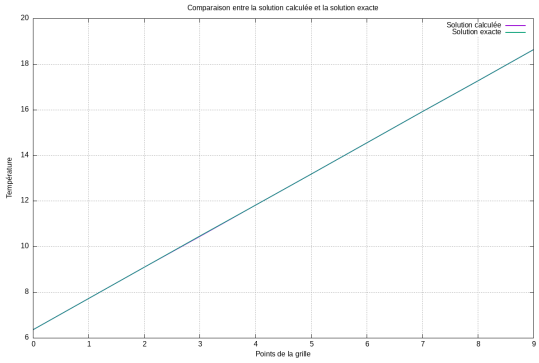

Figure 5: Comparaison de la solution exacte avec la solution calculée

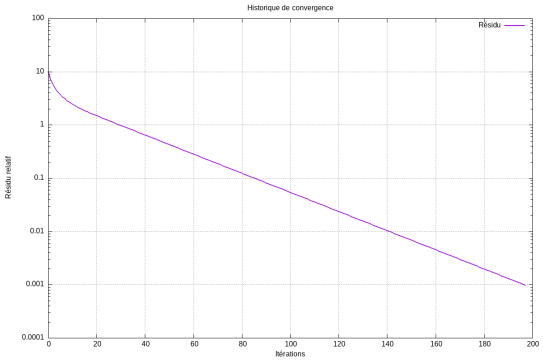

Figure 6: Convergence Richardson

6.4 Comparaison des 3 méthodes

On voit ici les différences de convergence entre les méthodes, on voit donc bien que les méthodes Jacobi et Gauss-Seidel, confondues sur le graphique, convergent moins vite que Richardson_alpha.

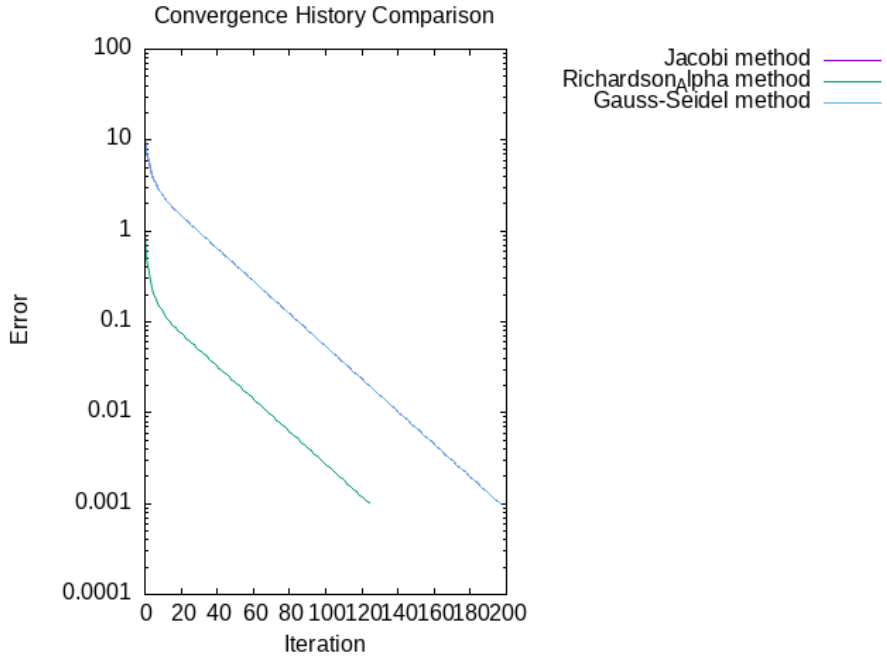

Figure 7: Comparaison des 3 méthodes

6.5 Stockage CSC et CSR, implémentation de Richardson

On voit ici que les méthodes CSC et CSR fonctionnent bien montrant une convergence équivalentes au méthodes avec matrices de préconditionnement. Elles utilisent l'implémentation de Richardson mais pour les matrices en rangement CSC/CSR. Erreur relative calculée : 2.489418e-04.

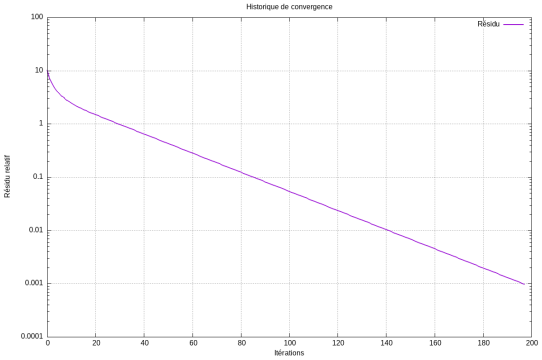

Figure 8: Convergence CSC

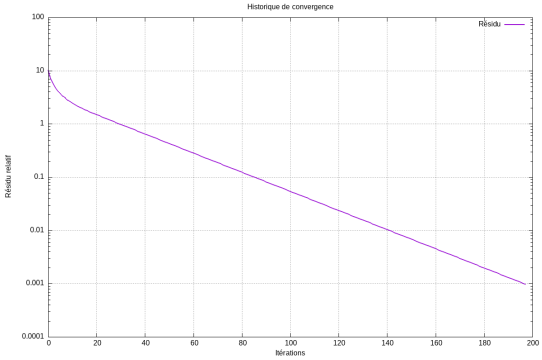

Figure 9: Convergence CSR