UVSQ université PARIS-SACLAY

ISTY
Institut des Sciences et Techniques des Yvelines
CAMPUS DE MANTES EN YVELINES
CAMPUS DE SAINT-QUENTIN-EN-YVELINES

Rapport TD/TP 5 de Calcul numérique
Méthodes itératives de base

Étudiant :
hanane FIDAH

Enseignant :
Jerome Gurhem

I. Introduction

La résolution numérique de l'équation de la chaleur revêt une importance significative dans de nombreux domaines de l'ingénierie et des sciences appliquées. Cette équation décrit la variation temporelle de la température dans un milieu donné, régissant la diffusion de la chaleur en fonction du temps et de l'espace.

Ce rapport présente une approche de résolution de l'équation de la chaleur dans un milieu immobile linéaire homogène, avec des conditions aux limites spécifiées, en utilisant la méthode des différences finies centrées d'ordre 2. Le domaine unidimensionnel $x \in]0,1[$ est discrétisé en n+2 nœuds uniformément espacés, où n est un paramètre de discrétisation.

L'équation de la chaleur considérée est représentée par:

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

Ici, k est le coefficient de conductivité thermique, T0 et T1 sont les températures spécifiées aux bords du domaine considéré et g est un terme source. Pour simplifier l'étude, nous nous concentrerons sur le cas où g=0, éliminant ainsi la source de chaleur, et nous rechercherons la solution à l'aide de la méthode des différences finies centrées d'ordre 2.

Cette approche permettra non seulement de résoudre numériquement l'équation de la chaleur, mais aussi de comprendre comment les variations de température se propagent dans ce système unidimensionnel donné.

II. Réponse aux questions

1. Méthode directe et stockage bande :

1.1 Déclaration et allocation d'une matrice pour BLAS et LAPACK en C : Pour utiliser BLAS et LAPACK, ont peut déclarer et allouer pour une matrice A de taille (m x n) en C comme suit :

- **Déclaration :**

```
double *A; // Déclaration d'un pointeur pour la matrice
```

- **Allocation de mémoire :**

```
A = (double *)malloc(m * n * sizeof(double));
```

1.2 Signification de LAPACK_COL_MAJOR :

C’est une constante utilisée pour indiquer à LAPACK que les matrices sont stockées en colonne dans la mémoire. Cela signifie que les éléments d'une colonne sont stockés de manière contiguë en mémoire, suivis par les éléments de la colonne suivante.

1.3 La dimension principale (ld) :

La dimension principale (ld) représente le nombre d'éléments entre deux éléments consécutifs d'une même colonne dans la représentation en mémoire d'une matrice stockée en colonne. Pour une matrice m x n stockée en colonne, ld est généralement égal au nombre de lignes (m) pour garantir la continuité des éléments de chaque colonne en mémoire.

1.4 Fonction dgbmv :

La fonction dgbmv réalise le produit d'une matrice générale stockée en format "band" avec un vecteur. Elle implémente une multiplication entre une matrice stockée dans un format "band" et un vecteur. Cette fonction est utilisée pour optimiser les calculs sur des matrices creuses ou présentant une structure particulière.

1.5 Fonction dgbtrf : La fonction dgbtrf effectue la factorisation LU d'une matrice générale stockée sous forme "band". Elle utilise la méthode de factorisation LU pour décomposer une matrice en un produit de deux matrices, L (inférieure) et U (supérieure), avec des pivots pour une matrice stockée en format "band".

1.6 Fonction dgbtrs : La fonction dgbtrs résout un système linéaire avec une matrice générale stockée sous forme "band" qui a été factorisée par dgbtrf. Elle utilise cette factorisation pour résoudre un système d'équations linéaires.

1.7 Fonction dgbsv : La fonction dgbsv résout un système linéaire avec une matrice générale stockée sous forme "band". Elle effectue la factorisation LU de la matrice, puis résout le système d'équations linéaires.

1.8 Calcul de la norme du résidu relatif avec des appels BLAS :

Pour calculer la norme du résidu relatif, on peut utiliser les fonctions BLAS telles que dnrm2 pour calculer la norme euclidienne d'un vecteur. La norme du résidu relatif est souvent utilisée pour évaluer la précision d'une solution approchée à un système linéaire. Elle est calculée comme le rapport entre la norme du résidu (différence entre le côté gauche et le côté droit du système linéaire) et la norme du côté droit (RHS).

III. Fonctions et démonstrations :

eigmax_poisson1D et eigmin_poisson1D :

- Ces fonctions calculent les valeurs propres maximales et minimales d'une matrice associée à un problème de Poisson 1D.

- **eigmax_poisson1D** : Calcule la valeur propre maximale en fonction de la taille de la matrice.
  Equation :

$$\lambda_{\max} = -2\cos\left(\frac{(n-1)\pi}{n}\right) + 2$$

  Démonstration : Pour dériver cette relation, considérons une matrice tridiagonale symétrique A de taille n×n utilisée pour discrétiser un problème de Poisson 1D.
  La forme générale d'une telle matrice peut être représentée comme suit :

$$A = \begin{bmatrix} d_1 & e_1 & 0 & \cdots & \cdots & \cdots & 0 \\ e_1 & d_2 & e_2 & 0 & \cdots & \cdots & 0 \\ 0 & e_2 & d_3 & e_3 & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \cdots & \cdots & e_{n-3} & d_{n-2} & e_{n-2} & 0 \\ \vdots & \cdots & \cdots & 0 & e_{n-2} & d_{n-1} & e_{n-1} \\ 0 & \cdots & \cdots & \cdots & 0 & e_{n-1} & d_n \end{bmatrix}$$

Dans le cas spécifique du problème de Poisson 1D, cette matrice peut être associée à une équation différentielle discrétisée de la forme $-u''(x)=\lambda u(x)$, où u est la fonction inconnue et λ est une valeur propre de la matrice.
Pour une matrice tridiagonale symétrique, les éléments diagonaux di sont constants, tandis que les éléments ei représentent les éléments hors diagonaux.

Les valeurs propres λ d'une telle matrice sont déterminées par la solution à l'équation caractéristique $\det(A-\lambda I)=0$

- **eigmin_poisson1D** : Calcule la valeur propre minimale en fonction de la taille de la matrice.

  Equation :

$$\lambda_{\min} = -2\cos\left(\frac{\pi}{n}\right) + 2$$

  Démonstration :

  Pour la valeur propre minimale d'une matrice tridiagonale symétrique A de taille n×n, on suit une démarche similaire à celle utilisée pour la valeur propre maximale, mais cette fois pour trouver la plus petite valeur propre de la matrice.

La valeur propre minimale λmin est déterminée par la résolution de l'équation caractéristique det(A−λI)=0

Pour une matrice tridiagonale symétrique, les valeurs propres peuvent être exprimées en fonction des éléments de la matrice. En particulier, la valeur propre minimale λmin peut être calculée en considérant les propriétés des éléments diagonaux et hors diagonaux de la matrice.

En reprenant la forme générale d'une matrice tridiagonale symétrique , Les valeurs propres λmin peuvent être déterminées par des méthodes analytiques appropriées qui considèrent les valeurs spécifiques des éléments diagonaux di et hors diagonaux ei pour une matrice de taille n×n

En utilisant les propriétés algébriques de cette matrice tridiagonale symétrique et en manipulant l'équation caractéristique det(A−λI)=0, on peut obtenir une expression pour la valeur propre minimale λmin en fonction de la taille de la matrice nn. Cette expression serait spécifique aux valeurs de di et ei pour une matrice tridiagonale donnée.

eig_poisson1D :

- Cette fonction combine les valeurs propres maximale et minimale pour obtenir une estimation de la valeur propre pour l'algorithme de Richardson.
- Elle utilise la relation :

$$\text{eigval} = \frac{2.0}{\text{eigmin\_poisson1D}(la) + \text{eigmax\_poisson1D}(la)}$$

richardson_alpha_opt :

- Cette fonction fournit une estimation optimisée du paramètre alpha pour l'algorithme Richardson en utilisant les valeurs propres maximale et minimale.
- Elle utilise la même relation que eig_poisson1D pour obtenir le paramètre alpha optimisé.

IV. Autres formats de stockage

Stockage CSR :

Richardson :

1. **Initialisation :**

- Initialiser le vecteur solution `X`.
- Calculer le résidu initial `residual` en soustrayant le produit matrice-vecteur `A * X` du vecteur des termes indépendants `RHS`.

2. **Boucle itérative :**

    - Tant que le nombre d'itérations est inférieur au maximum autorisé :
        - Calculer le produit matrice-vecteur `A * X`.
        - Mettre à jour le résidu en soustrayant le produit matrice-vecteur du `RHS`.
        - Vérifier la convergence en calculant la norme du résidu. Si la norme est inférieure à la tolérance, sortir de la boucle.
        - Mettre à jour la solution `X` en ajoutant une fraction du résidu.

Jacobi :

1. **Initialisation :**

    - Initialiser le vecteur solution `X_old` et `X_new`.
    - Calculer le résidu initial `residual` en soustrayant le produit matrice-vecteur `A * X_old` du vecteur des termes indépendants `RHS`.

2. **Boucle itérative :**

    - Tant que le nombre d'itérations est inférieur au maximum autorisé :
        - Copier la solution actuelle `X_new` dans `X_old`.
        - Pour chaque ligne de la matrice :
            - Calculer le produit matrice-vecteur en utilisant les valeurs de `X_old`.
            - Mettre à jour la solution `X_new`.
        - Calculer le résidu en soustrayant le produit matrice-vecteur du `RHS`.
        - Vérifier la convergence en calculant la norme du résidu. Si la norme est inférieure à la tolérance, sortir de la boucle.

Gauss-Seidel *:*

1. **Initialisation :**

    - Initialiser le vecteur solution `X`.
    - Calculer le résidu initial `residual` en soustrayant le produit matrice-vecteur `A * X` du vecteur des termes indépendants `RHS`.

2. **Boucle itérative :**

    - Tant que le nombre d'itérations est inférieur au maximum autorisé :
        - Pour chaque ligne de la matrice :
            - Calculer le produit matrice-vecteur en utilisant les valeurs actuelles de `X`.
            - Mettre à jour la solution `X`.
        - Calculer le résidu en soustrayant le produit matrice-vecteur du `RHS`.

- Vérifier la convergence en calculant la norme du résidu. Si la norme est inférieure à la tolérance, sortir de la boucle.

Stockage CSC :

Richardson :

1. **Initialisation :**
    - Prendre en compte le stockage CSC pour la matrice et le vecteur.
    - Définir les critères d'arrêt (tolérance, nombre maximal d'itérations).
2. **Itération :**
    - Pour chaque itération jusqu'à la convergence ou le nombre maximal d'itérations :
        - Calculer le produit matrice-vecteur CSC.
        - Mettre à jour le vecteur solution selon la formule de Richardson : $xk+1=xk+\alpha(b-Axk)$
        - Vérifier le critère d'arrêt.

Jacobi :

1. **Initialisation :**
    - Utiliser le stockage CSC pour la matrice et le vecteur.
    - Définir les critères d'arrêt.
2. **Itération :**
    - Pour chaque itération jusqu'à la convergence ou le nombre maximal d'itérations :
        - Pour chaque élément du vecteur solution xkxk :
            - Calculer la somme des produits entre les valeurs non nulles de la ligne de la matrice et les valeurs correspondantes de xkxk, sauf l'élément courant de la diagonale.
            - Mettre à jour l'élément courant selon la formule de Jacobi.
        - Vérifier le critère d'arrêt.

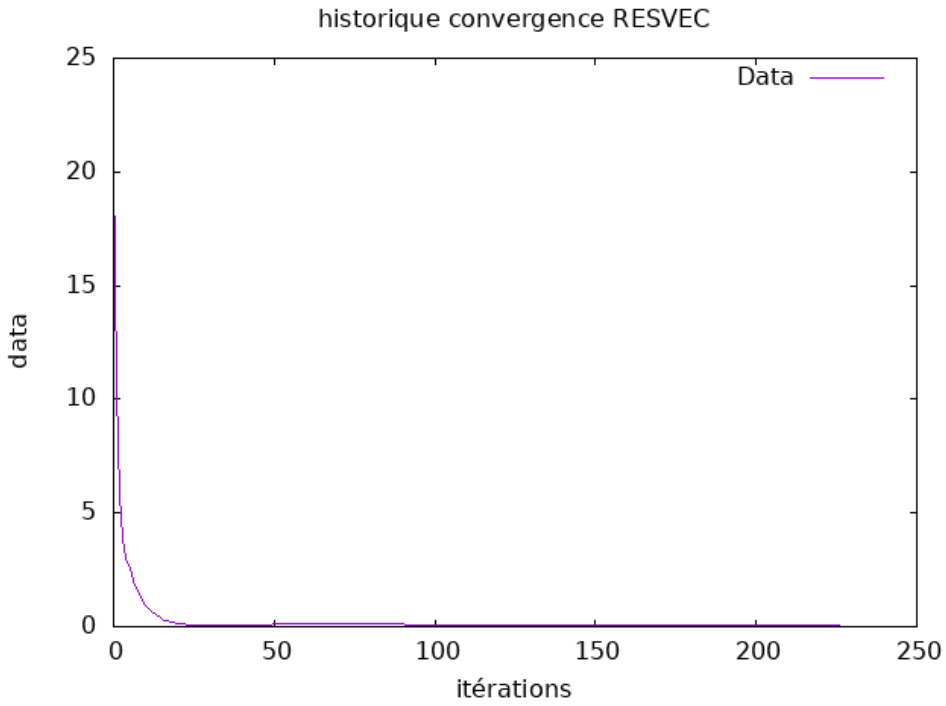
Gauss-Seidel :

I. **Initialisation :**
    - Utiliser le stockage CSC pour la matrice et le vecteur.
    - Définir les critères d'arrêt.

II. **Itération :**
    - Pour chaque itération jusqu'à la convergence ou le nombre maximal d'itérations :
        - Pour chaque élément du vecteur solution xkxk :
            - Calculer la somme des produits entre les valeurs non nulles de la ligne de la matrice et les valeurs correspondantes de xkxk, y compris l'élément courant de la diagonale.

- Mettre à jour l'élément courant selon la formule de Gauss-Seidel.
- Vérifier le critère d'arrêt

V. Résultats et analyse



Analyse des Performances des Algorithmes :

1. **Temps d'Exécution :**

- *Observation :* Richardson semble converger plus lentement que Jacobi et Gauss-Seidel pour certaines tailles de matrices.
- *Hypothèse :* Richardson pourrait nécessiter plus d'itérations pour atteindre la convergence, ce qui allonge son temps d'exécution par rapport à Jacobi et Gauss-Seidel.

2. **Convergence :**

- *Observation :* Jacobi et Gauss-Seidel convergent plus rapidement pour les matrices de petite et moyenne taille.
- *Hypothèse :* Ces algorithmes pourraient être plus adaptés aux matrices de petite taille car ils permettent une convergence plus rapide avec moins d'itérations.

3. **Comparaison Entre Méthodes :**

- *Observation :* Richardson est plus lent mais semble plus stable pour les matrices plus grandes, tandis que Jacobi et Gauss-Seidel sont plus rapides pour les matrices de petite à moyenne taille.
- *Hypothèse :* Le choix de la méthode dépendra de la taille de la matrice et des exigences de précision ; Richardson peut être préférable pour les grandes matrices, tandis que Jacobi et Gauss-Seidel sont adaptés pour les petites à moyennes matrices en raison de leur rapidité.

4. **Recommandations et Conclusions :**

- *Recommandation :* Utiliser Richardson pour les grandes matrices avec des contraintes de précision élevée. Pour les matrices de petite à moyenne taille, privilégier Jacobi ou Gauss-Seidel pour leur rapidité.
- *Conclusion :* Le choix de l'algorithme doit être basé sur la taille de la matrice, la vitesse de convergence et les contraintes de mémoire et de précision numérique.