TD/TP5 de Calcul numérique Méthodes itératives de base

Penda THIAO

17/12/2024

1 Exercice 1

Résolution de l'équation de la chaleur

Énoncé de l'exercice

On considère l'équation de la chaleur dans un milieu immobile, linéaire et homogène avec terme source et isotrope :

$$-k\frac{\partial^2 T}{\partial x^2} = g, \quad x \in]0,1[$$

avec les conditions aux bords :

$$T(0) = T_0, \quad T(1) = T_1$$

où g est un terme source, $k > 0$ est le coefficient de conductivité thermique, et $T_0 < T_1$ sont les températures aux bords du domaine considéré.

Nous allons résoudre cette équation par une méthode de différences finies centrée d'ordre 2.

Discrétisation et équation discrète

Le domaine $[0,1]$ est discrétisé selon $n+2$ noeuds x_i pour $i = 0, 1, 2, \ldots, n+1$, espacés d'un pas h constant. L'équation de la chaleur discrétisée en chaque noeud est obtenue par une approximation centrée de la dérivée seconde :

$$\frac{\partial^2 T}{\partial x^2} \approx \frac{T_{i+1} - 2T_i + T_{i-1}}{h^2}$$

Ainsi, l'équation discrète devient :

$$-k\frac{T_{i+1} - 2T_i + T_{i-1}}{h^2} = g_i$$

ou, en réarrangeant l'expression :

$$k\frac{T_{i+1} - 2T_i + T_{i-1}}{h^2} = g_i$$

Ce système peut être écrit sous la forme matricielle :

$$Au = f$$

où $A \in \mathbb{R}^{n \times n}$ est la matrice du système, u est le vecteur des températures discrètes T_i, et f est le vecteur des termes sources g_i.

Cas sans source de chaleur

Dans la suite du travail pratique, on considère le cas où il n'y a pas de source de chaleur, c'est-à-dire $g_i = 0$ pour tout i. Dans ce cas, l'équation discrétisée devient :

$$k\frac{T_{i+1} - 2T_i + T_{i-1}}{h^2} = 0$$

Ce qui peut être réarrangé pour donner une équation linéaire pour chaque T_i :

$$T_{i+1} - 2T_i + T_{i-1} = 0$$

Cela représente un système tridiagonal qui peut être résolu par des méthodes numériques appropriées.

Solution analytique

La solution analytique de l'équation de la chaleur sans terme source ($g = 0$) est donnée par la solution de l'équation différentielle :

$$-k\frac{\partial^2 T}{\partial x^2} = 0$$

La solution générale de cette équation est une fonction affine $T(x) = Ax + B$, où A et B sont des constantes à déterminer en utilisant les conditions aux bords. En appliquant les conditions $T(0) = T_0$ et $T(1) = T_1$, on obtient :

$$T(x) = T_0 + x(T_1 - T_0)$$

2 Exercice2

3 Exercice3

1. Déclaration et Allocation d'une Matrice en C pour BLAS et LAPACK

Pour utiliser BLAS et LAPACK en C, les matrices doivent être déclarées comme des tableaux en mémoire contiguë (1D), car ces bibliothèques s'attendent à des données linéarisées.

Exemple de déclaration et allocation :

```
int rows = 3;  // Nombre de lignes
int cols = 4;  // Nombre de colonnes

// Allouer un tableau 1D pour une matrice 3x4
double* matrix = (double*)malloc(rows * cols * sizeof(double));
```

Accès aux éléments :

```
// En supposant un stockage row-major :
matrix[i * cols + j] = valeur;  // i : ligne, j : colonne
```

Remarque : BLAS et LAPACK utilisent le format **colonne-major** par défaut, où les éléments des colonnes consécutives sont contigus en mémoire.

2. Signification de la Constante `LAPACK_COL_MAJOR`

La constante `LAPACK_COL_MAJOR` est utilisée dans l'interface C de LAPACK (`LAPACKE`) pour indiquer que les matrices sont stockées en **ordre colonne-major**, comme dans Fortran.

Stockage colonne-major :

- Les éléments d'une même colonne sont stockés consécutivement en mémoire.
- Exemple pour une matrice A 2×3 ($A[i][j]$) :

$$A = [1.0, 3.0, 2.0, 4.0, 5.0, 6.0]$$

Utilité : Cette constante garantit que la mémoire est organisée de manière compatible avec les routines LAPACK, qui utilisent par défaut le format colonne-major.

3. Dimension Principale (Leading Dimension, `ld`)

La **dimension principale** (`ld`) spécifie la longueur physique (en mémoire) d'une matrice, même si sa taille logique est plus petite.

Définition

La dimension principale correspond au **nombre d'éléments alloués par colonne** pour une matrice en format colonne-major. Pour une matrice A de taille $m \times n$, $ld \geq m$.

Exemple

Soit une matrice logique A 3×3, mais stockée dans un tableau 1D de taille 5×3. Ici :

- $ld = 5$ (nombre d'éléments dans chaque colonne).
- Stockage en mémoire :

$$[a_{11}, a_{21}, a_{31}, \_, \_, a_{12}, a_{22}, a_{32}, \_, \_, a_{13}, a_{23}, a_{33}, \_, \_]$$

Utilisation dans BLAS/LAPACK

Lorsque vous appelez une routine, vous devez fournir `ld` :

```
dgemm(LAPACK_COL_MAJOR, ..., A, ldA, ...);
```

4. Fonction `dgbmv`

La fonction `dgbmv` effectue une multiplication matrice-vecteur pour une matrice générale bande (band matrix).

Description

Elle calcule :

$$y = \alpha \cdot A \cdot x + \beta \cdot y$$

Où :

- A est une matrice bande, spécifiée par ses diagonales stockées en mémoire contiguë.
- x et y sont des vecteurs.
- α et β sont des scalaires.

Méthode

La méthode utilisée est une multiplication optimisée en tenant compte de la structure bande de la matrice.

5. Fonction `dgbtrf`

La fonction `dgbtrf` réalise la factorisation LU d'une matrice bande générale.

Description

Elle décompose une matrice A sous la forme :

$$A = P \cdot L \cdot U$$

Où :

- P est une matrice de permutation.
- L est une matrice triangulaire inférieure avec des 1 sur la diagonale.
- U est une matrice triangulaire supérieure.

Méthode

La méthode utilisée est une adaptation de l'algorithme LU pour les matrices bande, en réduisant les opérations inutiles sur les zéros implicites.

6. Fonction `dgbtrs`

La fonction `dgbtrs` résout un système linéaire utilisant la factorisation LU obtenue avec `dgbtrf`.

Description

Elle résout :

$$A \cdot x = b$$

Où A est déjà factorisée sous la forme LU avec `dgbtrf`.

Méthode

Elle utilise une substitution directe pour résoudre les systèmes triangulaires inférieurs et supérieurs.

7. Fonction `dgbsv`

La fonction `dgbsv` combine `dgbtrf` et `dgbtrs` pour résoudre un système linéaire en une seule étape.

Description

Elle résout :

$$A \cdot x = b$$

Où A est une matrice bande.

Méthode

Elle effectue :

1. La factorisation LU de A (`dgbtrf`).
2. La résolution du système factorisé (`dgbtrs`).

8. Calcul de la Norme du Résidu Relatif avec BLAS

Pour calculer la norme du résidu relatif, on peut utiliser les fonctions BLAS comme suit :

Étapes

1. Calculer le résidu $r = b - A \cdot x$ avec `dgemv` pour le produit matrice-vecteur.
2. Calculer la norme de r et celle de b avec `dnrm2` :

$$\text{norme relative} = \frac{\|r\|_2}{\|b\|_2}$$

4 Exercice 4. Stockage GB et appel à DGBMV

4.1 Question 1

```c
void set_GB_operator_colMajor_poisson1D(double* AB, int *lab, int *la, int *kv){
    int diagonals = *la; // Nombre de diagonales
    int cols = *lab;     // Nombre de colonnes

    //Ligne 0, que des zéros
    // Ligne 1 : Diagonale supérieure (-1)
    for (int j = 1; j < cols; j++) {
        AB[1 * cols + j] = -1.0;
    }

    // Ligne 2 : Diagonale principale (2)
    for (int j = 0; j < cols; j++) {
        AB[2 * cols + j] = 2.0;
    }

    // Ligne 3 : Diagonale inférieure (-1)
    for (int j = 0; j < cols - 1; j++) {
        AB[3 * cols + j] = -1.0;
    }
}
```

Stockage bande priorité colonne Poisson 1D

4.2 Question 2

4.3 Question 3

Pour valider notre stockage, on doit multiplier notre matrice AB par un vecteur unitaire $\mathbf{x} = (1, 1, 1, \ldots, 1)^T$. La matrice AB en stockage bande s'écrit :

$$AB = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & -1 \\ 2 & 2 & 2 & 2 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

Le vecteur $\mathbf{x}$ est donné par :

$$\mathbf{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

En effectuant le produit $\mathbf{y} = AB \cdot \mathbf{x}$, nous obtenons le vecteur attendu :

$$\mathbf{y} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Ce résultat valide que notre stockage en bande est correct.

5 Exercice 5

5.1 Résolution du systeme linéaire avec lapack

```c
/*Exercice 5*/
  /* It can also be solved with dgbsv */
  if (IMPLEM == SV) {
    // TODO : use dgbsv

    clock_t start, end;
    double time_used;
    start = clock();
    dgbsv_(&la,&kl,&ku,&NRHS,AB,&lab,ipiv,RHS,&la,&info);
    end = clock();

    time_used = ((double)(end-start))/CLOCKS_PER_SEC;
    printf("Time used = %f seconds\n",time_used);
  }
```

Resolution du system

Erreur Relative

```
  Solution with LAPACK

   INFO = 30

  The relative forward error is relres = 6.454972e-01
```

5.2 Evaluation des performances

Pour évaluer les performances des fonctions **DGBTRF**, **DGBTRS**, et **DGBSV**, on peut mesurer le temps d'exécution à l'aide de fonctions de chronométrage telles que `clock()` ou `gettimeofday()` en C.

Complexité théorique

Pour une matrice bande de taille $n \times n$, avec :

- kl : nombre de diagonales inférieures,
- ku : nombre de diagonales supérieures,

la complexité des opérations est la suivante :

1. **DGBTRF** (Factorisation LU pour matrice bande) :

$$\mathcal{O}(n \cdot kl \cdot ku)$$

Cette méthode tire profit de la structure bande pour réduire la complexité par rapport à une matrice dense.

2. **DGBTRS** (Résolution après factorisation LU) :

$$\mathcal{O}(n \cdot kl \cdot ku)$$

3. **DGBSV** (Factorisation et résolution combinées) :

$$\mathcal{O}(n \cdot kl \cdot ku)$$

C'est essentiellement une combinaison de **DGBTRF** et **DGBTRS**, donc la complexité reste identique.

Temps d'exécution

```
--------- End ----------
bin/tpPoisson1D_direct 2
--------- Poisson 1D ---------

Solution with LAPACK
Time used = 0.000248 seconds
```

6 Exercice 6

6.1 Implémentation de la méthode de factorisation LU

Soit une matrice tridiagonale A de dimension $n \times n$ avec une bande de largeur 3 (diagonale principale, diagonale supérieure et diagonale inférieure). La factorisation LU consiste à décomposer cette matrice en un produit de deux matrices triangulaires : une matrice triangulaire inférieure L et une matrice triangulaire supérieure U, telles que $A = LU$.
La matrice A est stockée dans le format GB (General Band), ce qui permet de représenter efficacement les matrices avec une bande de non-zéros.

Algorithme de la factorisation LU

L'algorithme de factorisation LU pour une matrice tridiagonale peut être implémenté de manière suivante :

- On parcourt chaque ligne de la matrice et on effectue les étapes suivantes pour chaque élément i de la diagonale principale :
  1. Si l'élément $AB[i + LAB]$ (pivot) est nul, on renvoie un code d'erreur.
  2. On calcule le facteur de pivot facteur $= \frac{AB[i+LAB+1]}{AB[i+LAB]}$.
  3. On met à jour la diagonale supérieure en effectuant l'élimination avec la relation $AB[i + LAB + 2] = AB[i + LAB + 2] - \text{facteur} \times AB[i + LAB + 1]$.
  4. La diagonale inférieure est modifiée pour stocker le facteur calculé.
- À la fin, la matrice A est transformée en une matrice triangulaire inférieure L et une matrice triangulaire supérieure U.

6.2 Méthode de validation

La validation de la factorisation LU peut être effectuée de deux manières principales :

Reconstruction de la matrice

La première méthode consiste à reconstruire la matrice A en multipliant les matrices L et U, et vérifier si le produit correspond à la matrice originale.

- On effectue la multiplication $A = L \times U$.
- On compare les éléments de la matrice reconstruite avec ceux de la matrice initiale et on vérifie si les écarts sont suffisamment faibles (en utilisant une tolérance).

Vérification des résultats

Une autre méthode consiste à comparer les résultats obtenus avec une bibliothèque éprouvée, comme LAPACK, pour effectuer la factorisation LU d'une matrice tridiagonale.