Université de Versailles Saint Quentin en Yvelines

Institut des Sciences et Techniques des Yvelines
M1 Calcul Haute Performance, Simulation (CHPS)

Rapport TP de calcul numérique :

Résolution de l’équation de la chaleur en 1D stationnaire

Élaboré par :

Abdellah DAHOUMANE

Année universitaire : 2022/2023

Le lien vers le dépôt Github du TP est :
https://github.com/abdahou/TP_CN_POISSON

1- Introduction :

Soit à résoudre l'équation de la chaleur dans un milieu immobile linéaire homogène avec terme source et isotrope :

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

Où g est un terme source, k > 0 le coefficient de conductivité thermique, et T0 < T1 les températures au bord du domaine considéré.
On se propose de résoudre cette équation par une méthode de différence finie centrée d'ordre 2. On discrétise le domaine 1D selon (N + 2) nœuds x_i , i = 0, 1, 2, ...N + 1, espacés d'un pas h constant.
En chaque nœud l'équation discrète s'écrit :

$$-k\left(\frac{\partial^2 T}{\partial x^2}\right)_i = g_i$$

Pour la suite du TP on notera ce système :

$$Au = f, \ A \in \mathbb{R}^{n\times n}, \ u, \ f \in \mathbb{R}^n$$

Dans la suite du TP on considère qu'il n'y a pas de source de chaleur, i.e. g = 0. La solution analytique de cette équation est :

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

1. Approximation de la dérivée seconde de T au moyen d'un schéma centré d'ordre 2 :

En utilisant le développement de Taylor à l'ordre 2, on a :

$$\begin{cases} T(x+h) = T(x) + h\frac{\partial T}{\partial x} + \frac{h^2}{2}\frac{\partial^2 T}{\partial x^2} + O(h^2) \\ T(x-h) = T(x) - h\frac{\partial T}{\partial x} + \frac{h^2}{2}\frac{\partial^2 T}{\partial x^2} + O(h^2) \end{cases}$$

En sommant les deux équations ci-dessus, on obtient :

$$T(x+h) + T(x-h) = 2T(x) + h^2\frac{\partial^2 T}{\partial x^2} + O(h^2)$$

D'où l'approximation de la dérivée seconde de T au moyen d'un schéma centré d'ordre 2 s'écrit :

$$-\frac{\partial^2 T}{\partial x^2} = \frac{-T(x-h) + 2T(x) - T(x+h)}{h^2}$$

2. Écriture le système linéaire de dimension n correspondant au problème 1 :

Au nœuds x_i de la discrétisation on a :

$$\begin{cases} T(x_i) = U_i \\ A \times U = f \end{cases}$$

avec :

$$\begin{cases} T(x_0) = T_0 = U_0 \\ T(x_{N+1}) = T(1) = T_1 = U_{N+1} \end{cases}$$

Comme le système est discrétisé en (N + 2) nœuds espacés d'un pas h constant, donc :

$$x_{N+1} = (N+1) \times h$$

D'où :

$$\boxed{h = \frac{1}{N+1}}$$

Le schéma centré s'écrit alors :

$$k\ \frac{-U_{i-1} + 2U_i - U_{i+1}}{h^2} = g_i, \quad \forall i \in [1, n]$$

Pour i=1 :

$$\begin{cases} k\ \frac{-U_2 + 2U_1 - U_0}{h^2} = g \\ U_0 = T_0 \end{cases}$$

On obtient donc :

$$2U_1 - U_2 = \frac{h^2}{k} g_1 + T_0$$

Pour i=2 :

$$k\ \frac{-U_1 + 2U_2 - U_3}{h^2} = g_2$$

D'où :

$$-U_1 + 2U_2 - U_3 = \frac{h^2}{k} g_2$$

Pour i=N :

$$\begin{cases} k\ \frac{-U_{N-1} + 2U_N - U_{N+1}}{h^2} = g_N \\ T(x_{N+1}) = T_1 = U_{N+1} \end{cases}$$

On obtient donc :

$$-U_{N-1} + 2U_N = \frac{h^2}{k} g_N + T_1$$

D'après les équations obtenues le système linéaire A.u=f de dimension n correspondant au problème s'écrit :

$$\begin{bmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & -1 & 2 & -1 & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \dots & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_{n-1} \\ U_n \end{bmatrix} = \begin{bmatrix} \frac{h^2}{k} g_1 + T_0 \\ \frac{h^2}{k} g_2 \\ \vdots \\ \frac{h^2}{k} g_{N-1} \\ \frac{h^2}{k} g_N + T_1 \end{bmatrix}$$

3. Le problème à résoudre et sa discrétisation :

$$Au = f,\ A \in \mathbb{R}^{n \times n},\ u,\ f \in \mathbb{R}^n$$

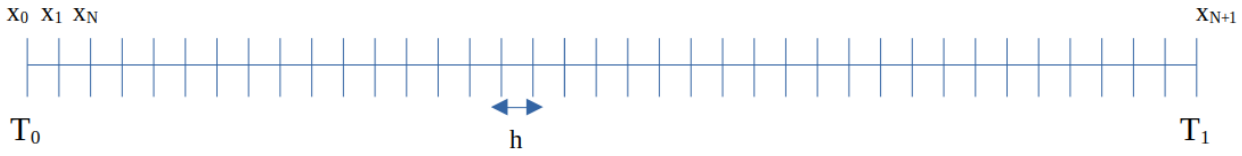

Figure 1: Discrétisation du problème

Exercice 2: Mise en place de l'environnement de développement (C + BLAS/LAPACK)

Travail fait et à trouver sur le dépôt Github du projet : https://github.com/abdahou/TP_CN_POISSON

2- Méthode directe et stockage bande :

On considère dans notre étude des matrices tridiagonales. Une méthode de stockage efficace pour ce type de matrice est le stockage par bande.

Lors de l'utilisation de LAPACK et BLAS, différents stockages bande sont proposés. Nous étudions dans un premier temps le stockage General Band.

Une matrice de dimension m × n avec kl sous-diagonales et ku sur-diagonales peut être stockée de manière compacte dans un tableau à deux dimensions avec kl + ku + 1 lignes et n colonnes.. Ce stockage ne doit être utilisé que si kl,ku<< min(m, n). Dans LAPACK les matrices ayant un stockage de ce type ont un nom se terminant par B. Il suit que l'élément a_{ij} de la matrice A est stockée dans AB(ku + 1 + i − j, j). Par exemple, pour m = 5, n = 5, kl = 2, ku = 1; Soit :

$$A = \begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & a_{42} & a_{43} & a_{44} & a_{45} \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix}.$$

sont stockage GB dans le tableau AB est tel que :

$$AB = \begin{pmatrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{pmatrix}.$$

Où :
- Les colonnes de la matrice sont stockées dans les colonnes correspondantes du tableau.
- Les diagonales de la matrice sont stockées dans les lignes du tableau.

Dans la suite du TP, on va étudier le code C associé au TP, en faisant appel à LAPACK et BLAS.

Exercice 3: Référence et utilisation de BLAS/LAPACK

1. Pour déclarer et allouer une matrice en C pour utiliser BLAS et LAPACK, on utilise la fonction "malloc" pour allouer de l'espace mémoire pour la matrice, puis on utilise un pointeur pour y accéder.
Comme indiquer sur l'exemple suivant :

```
AB = (double *) malloc(sizeof(double)*lab*la);
```

Où :
« lab » et « la » sont respectivement les nombres de lignes et de colonnes dans la matrice,

2. **LAPACK_COL_MAJOR** est une constante utilisée pour spécifier l'ordre de stockage des données dans une matrice en utilisant la bibliothèque de calcul scientifique LAPACK. La valeur de cette constante est 1 et indique que les données sont stockées en colonne-principale, c'est-à-dire que les éléments d'une colonne sont stockés les uns à côté des autres en mémoire. Cela est en contraste avec l'ordre de stockage ligne-principale, où les éléments d'une ligne sont stockés les uns à côté des autres.

3. La dimension principale (leading dimension) généralement notée ld est utilisée pour spécifier l'espacement entre les éléments consécutifs d'une matrice en mémoire. Elle est souvent utilisée dans les bibliothèques de calcul scientifique LAPACK pour permettre aux matrices de disposer de suffisamment d'espace pour stocker des sous-matrices ou des extensions.

Pour une matrice en format colonne-majeur, la dimension principale est la distance (mesurée en nombre d'éléments) entre le premier élément de chaque colonne de la matrice en mémoire. Pour une matrice en format ligne-majeur, la dimension principale est la distance entre le premier élément de chaque ligne de la matrice en mémoire.

4. La fonction **dgbmv** (Double-precision General Banded Matrix-Vector product) est une fonction qui permet de faire le produit matrice-vecteur entre une matrice bandée générale GB et un vecteur. Elle calcule (y = alpha * A * x + beta * y), où A est une matrice bandée générale avec kl diagonales inférieures et ku diagonales supérieures, x est le vecteur entrée, y est le vecteur de sortie, alpha et beta sont des scalaires.

La fonction dgbmv prend en entrée les paramètres suivants :

- trans: un caractère qui spécifie si la matrice doit être transposée ou non (c'est-à-dire si le produit matrice-vecteur doit être A * x ou A^T * x)
- m: la taille de la matrice (nombre de lignes)
- n: la taille de la matrice (nombre de colonnes)
- kl: le nombre de diagonales inférieures
- ku: le nombre de diagonales supérieures
- alpha: un scalaire qui multiplie la matrice
- a: la matrice bandée générale stockée en colonne-majeur
- lda: la dimension principale (leading dimension) de la matrice a
- x: le vecteur entrée
- incx: le pas d'incrément pour parcourir le vecteur x
- beta: un scalaire qui multiplie le vecteur de sortie
- y: le vecteur de sortie
- incy: le pas d'incrément pour parcourir le vecteur y

- La fonction **dgbmv** de LAPACK implémente la méthode du produit matrice-vecteur traditionnel pour calculer le produit matrice-vecteur entre une matrice bandée générale et un vecteur.

5. La fonction **dgbtrf** (Double-precision General Banded Matrix LU factorization) est une fonction qui permet de calculer la factorisation LU d'une matrice bandée générale. Elle permet de décomposer une matrice bandée générale A en deux matrices triangulaires L et U de telle sorte que A = L * U.

La fonction dgbtrf prend en entrée les paramètres suivants :

- m: le nombre de lignes de la matrice A
- n: le nombre de colonnes de la matrice A
- kl: le nombre de diagonales inférieures de la matrice A
- ku: le nombre de diagonales supérieures de la matrice A
- a: la matrice bandée générale A, stockée en colonne-majeur
- lda: la dimension principale (leading dimension) de la matrice A
- ipiv: un tableau qui contient les permutations de lignes effectuées par la fonction pour rendre la matrice A échelonnée.

- La fonction **dgbtrf** implémente une méthode de pivot de Gauss pour obtenir la factorisation LU. Elle utilise les propriétés spécifiques de la matrice bandée générale pour optimiser les calculs en n'utilisant que les éléments non-nuls de la matrice. La fonction dgbtrf remplit la matrice A avec les matrices L et U. Elle remplit également le tableau ipiv avec les permutations de lignes effectuées pour obtenir la factorisation LU.

6. La fonction **dgbtrs** (Double-precision General Banded Matrix triangular Solver) est une fonction de la bibliothèque LAPACK qui permet de résoudre un système linéaire Ax = b avec une matrice bandée générale A. Elle utilise la factorisation LU de la matrice A obtenue par la fonction **dgbtrf** pour résoudre le système linéaire.

La fonction dgbtrs prend en entrée les paramètres suivants :

- trans: un caractère qui spécifie si la matrice doit être transposée ou non (c'est-à-dire si le système linéaire doit être résolu pour A * x = b ou A^T * x = b)
- n: le nombre de colonnes de la matrice A
- kl: le nombre de diagonales inférieures de la matrice A
- ku: le nombre de diagonales supérieures de la matrice A
- nrhs: le nombre de vecteurs b dans le système linéaire (nombre de colonnes de la matrice de droite)
- a: la matrice bandée générale A, stockée en colonne-majeur, contenant les matrices L et U obtenues par la factorisation LU
- lda: la dimension principale (leading dimension) de la matrice A
- ipiv: un tableau qui contient les permutations de lignes effectuées par la fonction dgbtrf pour rendre la matrice A échelonnée
- b: le vecteur ou la matrice de droite du système linéaire
- ldb: la dimension principale (leading dimension) de la matrice b

- La fonction **dgbtrs** implémente les matrices L et U obtenues par la factorisation LU de la matrice A pour résoudre le système linéaire en utilisant des substitutions successives. Elle utilise également les permutations de lignes effectuées par la fonction dgbtrf pour rendre la matrice A échelonnée.

7. La fonction **dgbsv** (Double-precision General Banded Matrix Solver) est une fonction de la bibliothèque LAPACK qui permet de résoudre un système linéaire Ax = B avec une matrice bandée générale A. Elle utilise la factorisation LU pour résoudre le système linéaire de manière efficace.

La fonction dgbsv prend en entrée les paramètres suivants :

- matrix Layout (int) : indique si les matrices sont stockées dans l'ordre principal des lignes (matrix layout = LAPACK ROW MAJOR) ou l'ordre principal des colonnes (matrix layout = LAPACK COL MAJOR)
- n : indique l'ordre de la matrice A et le nombre de lignes de la matrice B. Dans notre cas, elle est évaluée à "la"
- kl : indique la largeur de la bande inférieure (sous-diagonale)
- ku : indique la largeur de la bande supérieure (sur-diagonale)
- NRHS : indique le nombre de colonnes de la matrice B
- AB : indique le tableau 2D de taille lab × la dans lequel on a stocké la matrice bande
- ldab : indique la dimension principale de AB égale à "la" dans le cas où "matrix layout = LAPACK ROW MAJOR" et à "lab" dans le cas où "matrix layout = LAPACK COL MAJOR"
- ipiv : indique les éléments contenants les indices du pivot. Il s'agit d'un vecteur de longeur "la"
- b : indique la matrice générale B, contenant les nrhs membres droits du système. Dans notre cas, elle est évaluée à nrhs
- ldb : indique la dimension principale de B. Dans notre cas, elle est évaluée à nrhs

- La fonction **dgbsv** implémente une méthode de pivot de Gauss pour obtenir la factorisation LU de la matrice A. Elle utilise également les permutations de lignes effectuées pour rendre la matrice A échelonnée pour résoudre le système linéaire en utilisant des substitutions successives. La fonction remplit la matrice B avec la solution x du système.

8. Pour calculer la norme du résidu relatif d'un système linéaire Ax = B, on utilise les fonctions BLAS pour calculer la norme de la différence entre le vecteur B et le produit de la matrice A par la solution x. La norme du résidu relatif est définie comme :

$$||B - Ax|| / ||B||$$

Pour calculer cette valeur, on utilise les fonctions BLAS suivantes :

- dgbmv (ou dgemv si A est dense) pour calculer le produit matrice-vecteur Ax.
- daxpy pour calculer la différence B – Ax.
- dnrm2 pour calculer la norme de la différence B – Ax.
- dnrm2 pour calculer la norme de B.

Exercice 4: Stockage GB et appel à DGBMV

1. Ecriture du stockage GB en priorité colonne pour la matrice de Poisson 1D :

```
 9  void set_GB_operator_colMajor_poisson1D(double* AB, int *lab, int *la, int *kv){
10    int ii, jj, kk;
11    for (jj=0;jj<(*la);jj++){
12      kk = jj*(*lab);
13      if (*kv>=0){
14        for (ii=0;ii< *kv;ii++){
15    AB[kk+ii]=0.0;
16        }
17      }
18      AB[kk+ *kv]=-1.0;
19      AB[kk+ *kv+1]=2.0;
20      AB[kk+ *kv+2]=-1.0;
21    }
22    AB[0]=0.0;
23    if (*kv == 1) {AB[1]=0;}
24
25    AB[(*lab)*(*la)-1]=0.0;
26  }
```

2. Utilisation de la fonction BLAS dgbmv :

```
60  write_GB_operator_colMajor_poisson1D(AB, &lab, &la, "AB.dat");
61
62    cblas_dgbmv(CblasColMajor, CblasConjNoTrans, la, la, kl, ku, 1.0, AB+1, lab, EX_SOL, 1, 0.0, RHS_blas, 1);
63    write_vec(RHS_blas, &la, "dgbvm.dat");
```

3. Proposition d’une méthode de validation :

Pour valider cette de méthode de résolution par la fonction **dgbmv** on calcul l’erreur relative :

$$relres = \frac{\|SOL\_Exacte - Y\|}{\|Y\|}$$

Où on trouve après compilation une erreur égale à : 2.692638e-16 qui est une erreur très faible ce qui nous permet de valider la méthode.

Exercice 5: DGBTRF, DGBTRS, DGBSV

1. Résolution du système linéaire par une méthode directe en faisant appel à LAPACK :

```
  //******LAPACKE_dgbsv**********//
set_dense_RHS_DBC_1D(RHS,&la,&T0,&T1);
info = LAPACKE_dgbsv(LAPACK_COL_MAJOR, la, kl, ku, NRHS, AB, lab, ipiv, RHS, la);

if (info == 0) {
  printf("\n INFO DGBSV = %d\n",info);
  printf("LAPACKE_dgbsv SOL]\n\n");
  for (int i = 0; i < la; i++) {
    printf("X[%d] = %f\n", i, RHS[i]);
  }
} else {
  printf("Erreur : le système d'équations n'a pas pu être résolu.\n");
}
//
```

```
//******LAPACKE_dgbtrs**********//

set_dense_RHS_DBC_1D(RHS,&la,&T0,&T1);
info = LAPACKE_dgbtrs(LAPACK_COL_MAJOR, 'N', la, kl, ku, NRHS, AB, lab, ipiv, RHS, la);
if (info == 0) {
  printf("LAPACKE_dgbtrs SOL]\n\n");
  for (int i = 0; i < la; i++) {
    printf("X[%d] = %f\n", i, RHS[i]);
  }
} else {
  printf("Erreur : le système d'équations n'a pas pu être résolu.\n");
}
```

Exercice 6: LU pour les matrices tridiagonales

1. Implémentation la méthode de factorisation LU pour les matrices tridiagonales avec le format GB :

```
// Factorisation LU
void LU_Facto(double* AB, int *lab, int *la, int *kv){
    int i, j, k, diags = 3;


      if (*kv>=0){
        diags = 4;
        for (i=0;i< *kv;i++){
            AB[i]=0.0;
        }
      }
      AB[*kv+2]/=AB[*kv+1];


    for (j=1;j<(*la);j++){
      k = j*(*lab);
      if (*kv>=0){
        for (i=0;i< *kv;i++){
            AB[k+i]=0.0;
        }
      }

      AB[k+ *kv+1]-=AB[k+ *kv]*AB[(k-diags)+ *kv+2];
      AB[k+ *kv+2]/=AB[k+ *kv+1];
    }

}
```

2- Méthode de résolution itérative

Nous souhaitons maintenant résoudre l'équation de la chaleur par une méthode itérative. Dans les exercices suivant nous étudions différents algorithmes et leur conception pour une matrice tridiagonale et plus particulièrement pour ce problème. Dans un premier temps on effectue une étude théorique puis un maquettage avec Scilab. Dans un second temps on développe le code C reposant sur l'appel des BLAS.

Exercice 7: Implémentation C - Richardson

1. implanter en C l'algorithme de Richardson avec des matrices au format GB. Sauvegardez le résidu dans un vecteur :

l'implémentation de l'algorithme de Richardson avec des matrices au format GB se trouve dans le code source « tp_poisson1D_iter.c » au dossier «src » du projet.

Références :

[1] Matrix storage scheme: http://www.netlib.org/lapack/lug/node121.html
[2] Band Storage: http://www.netlib.org/lapack/lug/node124.html
[3] BLAS Documentation:https://netlib.org/blas/
[4] LAPACK Documentation: http://www.netlib.org/lapack
[5] The LAPACKE C Interface to LAPACK: http://www.netlib.org/lapack/lapacke
[6] CLAPACK The Fortran to C version of LAPACK:https://netlib.org/clapack/