



Université de Versailles Saint-Quentin-en-Yvelines  
Master 1 CHPS – Calcul Numérique 2025/2026

---

# Rapport de TP : Méthodes Directes et Itératives pour l'Équation de la Chaleur 1D

---

*Auteur:*

Jianye SHI

*Enseignant:*

T. Dufaud, J. Gurhem

## Sommaire

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Problème Physique . . . . .	3
1.2	Discrétisation par Différences Finies . . . . .	3
1.3	Solution Analytique (Cas $g = 0$ ) . . . . .	3
<b>2</b>	<b>Environnement et Stockage</b>	<b>3</b>
2.1	Environnement de Développement . . . . .	3
2.2	Stockage en bande (General Band, GB) . . . . .	4
<b>3</b>	<b>Méthode Directe : Factorisation LU</b>	<b>4</b>
3.1	Utilisation de LAPACK . . . . .	4
3.2	Implémentation LU Tridiagonale . . . . .	4
3.3	Complexité . . . . .	4
<b>4</b>	<b>Méthodes Itératives</b>	<b>5</b>
4.1	Méthode de Richardson . . . . .	5
4.2	Méthode de Jacobi . . . . .	5
4.3	Méthode de Gauss-Seidel . . . . .	5
4.4	Critère de Convergence . . . . .	5
<b>5</b>	<b>Validation et Résultats</b>	<b>6</b>
5.1	Méthode de Validation . . . . .	6
5.2	Résultats Numériques . . . . .	6
5.3	Temps d'Exécution . . . . .	6
5.4	Convergence des Méthodes Itératives . . . . .	7
<b>6</b>	<b>Formats de stockage creux (CSR/CSC)</b>	<b>8</b>
6.1	Implémentation des Structures . . . . .	8
6.2	Produit Matrice-Vecteur Creux (SpMV) . . . . .	8
6.3	Adaptation des Algorithmes Itératifs . . . . .	9
6.4	Validation . . . . .	9
<b>7</b>	<b>Conclusion</b>	<b>9</b>

## 1 Introduction

Ce rapport présente la réalisation du TP sur les **méthodes directes et itératives** pour la résolution de l'équation de la chaleur 1D stationnaire. L'objectif est d'appliquer les algorithmes vus en cours/TD pour résoudre un système linéaire issu de la discréétisation par différences finies.

### 1.1 Problème Physique

On considère l'équation de la chaleur 1D stationnaire dans un milieu immobile, linéaire, homogène et isotrope, avec terme source :

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

où  $g$  est le terme source,  $k > 0$  la conductivité thermique, et  $T_0 < T_1$  les températures aux bords (conditions de Dirichlet).

### 1.2 Discréétisation par Différences Finies

On discréétise le domaine  $[0,1]$  sur  $n+2$  nœuds  $x_i$ ,  $i = 0, \dots, n+1$ , avec un pas constant  $h = \frac{1}{n+1}$ . En appliquant un schéma centré d'ordre 2 pour approximer la dérivée seconde :

$$\left( \frac{\partial^2 T}{\partial x^2} \right)_i \approx \frac{T_{i-1} - 2T_i + T_{i+1}}{h^2} \quad (2)$$

En multipliant chaque équation par  $-h^2/k$ , on obtient le système linéaire  $Au = f$ , où  $u = (T_1, \dots, T_n)^T$  est le vecteur des températures aux nœuds internes. Le vecteur  $f$  absorbe le terme source  $g_i$  et les conditions aux limites, mis à l'échelle par  $h^2/k$ .

On obtient un système linéaire de dimension  $n$  :

$$Au = f, \quad A \in \mathbb{R}^{n \times n}, \quad u, f \in \mathbb{R}^n \quad (3)$$

où  $A$  est une matrice tridiagonale de la forme :

$$A = \begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & -1 & 2 & -1 & \\ & & & -1 & 2 & \\ & & & & -1 & 2 \end{pmatrix} \quad (4)$$

### 1.3 Solution Analytique (Cas $g = 0$ )

Dans le cas sans source ( $g = 0$ ), la solution analytique est linéaire :

$$T(x) = T_0 + x(T_1 - T_0) \quad (5)$$

Cette solution sert de référence pour valider nos implémentations.

## 2 Environnement et Stockage

### 2.1 Environnement de Développement

Le code est développé en **langage C** avec les bibliothèques **BLAS** et **LAPACK**. L'environnement est configuré et exécuté exclusivement via **Docker** pour garantir la portabilité et la reproductibilité des résultats, conformément aux instructions (fichier `README.md`).

## 2.2 Stockage en bande (General Band, GB)

Pour les matrices tridiagonales, on utilise le format **General Band** (GB) de LAPACK. Une matrice  $m \times n$  avec  $kl$  sous-diagonales et  $ku$  sur-diagonales se stocke dans un tableau  $(kl + ku + 1) \times n$ . L'élément  $a_{ij}$  est stocké en position  $AB(ku + 1 + i - j, j)$ .

**Exemple** pour la matrice de Poisson 1D ( $kl = ku = 1$ ) :  $AB$  représente la matrice  $A$  sous forme bande (General Band), où  $A$  est tridiagonale.

$$AB = \begin{pmatrix} 0 & -1 & -1 & \cdots & -1 \\ 2 & 2 & 2 & \cdots & 2 \\ -1 & -1 & -1 & \cdots & 0 \end{pmatrix} \quad (6)$$

L'implémentation utilise la fonction `set_GB_operator_colMajor_poisson1D` :

```
AB = zeros(lab, n);
% Remplissage (Tridiagonale -1, 2, -1)
AB(kv+1, 2:n) = -1.0; % Sur-diagonale
AB(kv+2, 1:n) = 2.0; % Diagonale
AB(kv+3, 1:n-1) = -1.0; % Sous-diagonale
```

Validation : comparer le résultat de `dgbmv` avec un produit dense de référence (ou une solution analytique) sur un petit cas test.

## 3 Méthode Directe : Factorisation LU

### 3.1 Utilisation de LAPACK

Nous utilisons les routines LAPACK pour la résolution directe :

- `dgbtrf` : Factorisation LU avec pivotage partiel
- `dgbtrs` : Résolution triangulaire après factorisation
- `dgbsv` : Driver combinant factorisation et résolution

### 3.2 Implémentation LU Tridiagonale

Pour les matrices tridiagonales, une factorisation LU simplifiée (sans pivotage) a été implémentée dans `dgbtrftridiag` :

```
for j = 1:n-1
    % Element diagonal courant
    pivot = AB(diag_row, j);

    % Facteur pour éliminer l'élément sous-diagonal (L)
    factor = AB(sub_row, j) / pivot;
    AB(sub_row, j) = factor;

    % Mise à jour de l'élément diagonal suivant (U)
    AB(diag_row, j+1) = AB(diag_row, j+1) - factor * AB(sup_row, j+1);
end
```

### 3.3 Complexité

- `dgbtrf` :  $O(n(kl + ku)^2) = O(n)$  pour tridiagonal
- `dgbtrs` :  $O(n(kl + ku)) = O(n)$  pour tridiagonal

## 4 Méthodes Itératives

### 4.1 Méthode de Richardson

La méthode de Richardson avec paramètre de relaxation  $\alpha$  s'écrit :

$$x^{(k+1)} = x^{(k)} + \alpha r^{(k)}, \quad r^{(k)} = b - Ax^{(k)} \quad (7)$$

Le paramètre optimal est :

$$\alpha_{opt} = \frac{2}{\lambda_{min} + \lambda_{max}} \quad (8)$$

où les valeurs propres de la matrice de Poisson 1D sont :

$$\lambda_k = 2 - 2 \cos \left( \frac{k\pi}{n+1} \right), \quad k = 1, \dots, n \quad (9)$$

### 4.2 Méthode de Jacobi

La méthode de Jacobi utilise le préconditionneur  $M = D$  (diagonale de  $A$ ) :

$$x^{(k+1)} = x^{(k)} + D^{-1}r^{(k)} \quad (10)$$

L'extraction du préconditionneur :

```
MB = zeros(lab, n);
for j = 1:n
    MB(kv+2, j) = AB(ku+1, j); % Extraction Diagonale
end
```

### 4.3 Méthode de Gauss-Seidel

Gauss-Seidel utilise  $M = D - E$  (partie triangulaire inférieure) :

$$x^{(k+1)} = x^{(k)} + (D - E)^{-1}r^{(k)} \quad (11)$$

La résolution de  $(D - E)z = r$  se fait par substitution avant.

### 4.4 Critère de Convergence

Les démonstrations des conditions de convergence et du choix optimal de  $\alpha$  ont été vues en cours et sont détaillées dans les TD associés.

La convergence est vérifiée par le résidu relatif :

$$\frac{\|r^{(k)}\|}{\|b\|} < \varepsilon \quad (12)$$

avec  $\varepsilon = 10^{-3}$  par défaut.

Chaque itération coûte  $O(n)$  pour une matrice tridiagonale ; le coût total est donc proportionnel au nombre d'itérations.

## 5 Validation et Résultats

### 5.1 Méthode de Validation

La validation s'effectue par :

1. Comparaison avec la solution analytique (erreur relative avant)
2. Calcul du résidu relatif  $\|Ax - b\|/\|b\|$
3. Comparaison avec les résultats LAPACK de référence

L'erreur relative avant est calculée par :

```
function err = relative_forward_error(x, y)
    err = norm(y - x) / norm(x);
end
```

### 5.2 Résultats Numériques

Les tests ont été réalisés avec  $T_0 = -5$ ,  $T_1 = 5$ , et différentes tailles de problème.

Méthode	N=10 (Err)	N=100
LU (Tridiag)	$< 10^{-14}$	$< 10^{-14}$
Richardson	$3.8 \times 10^{-3}$	$> 1000$
Jacobi	$3.8 \times 10^{-3}$	$> 1000$
Gauss-Seidel	$3.8 \times 10^{-3}$	$> 1000$

TABLE 1 – Erreur relative avant ( $\|x_{calc} - x_{ex}\|/\|x_{ex}\|$ )

**Analyse :** Pour  $N = 10$ , l'erreur correspond à la discréétisation ( $O(h^2)$ ). Pour  $N \geq 100$ , les méthodes itératives n'ont pas atteint la tolérance en 1000 itérations (nombre maximal d'itérations, 1000), bien que théoriquement convergentes. La lenteur s'explique par le conditionnement qui croît comme  $O(N^2)$ .

### 5.3 Temps d'Exécution

*Environnement de test : exécution dans un conteneur Docker sur Fedora Linux 42. Hôte : AMD Ryzen 9 8940HX (16 cœurs), 30 GiB RAM. Compilation GCC avec optimisation -O2.*

Méthode	N=100	N=1000	N=10000
Direct (Tridiag)	0.05 ms	0.07 ms	0.56 ms
Direct (LAPACK)	0.03 ms	0.08 ms	0.57 ms

TABLE 2 – Temps de calcul (Méthodes Directes)

La Figure 1 montre l'évolution du temps de calcul en fonction de la taille de la matrice  $N$ . On observe une croissance linéaire ( $O(N)$ ), ce qui valide l'efficacité des solveurs bandes pour ce problème 1D.

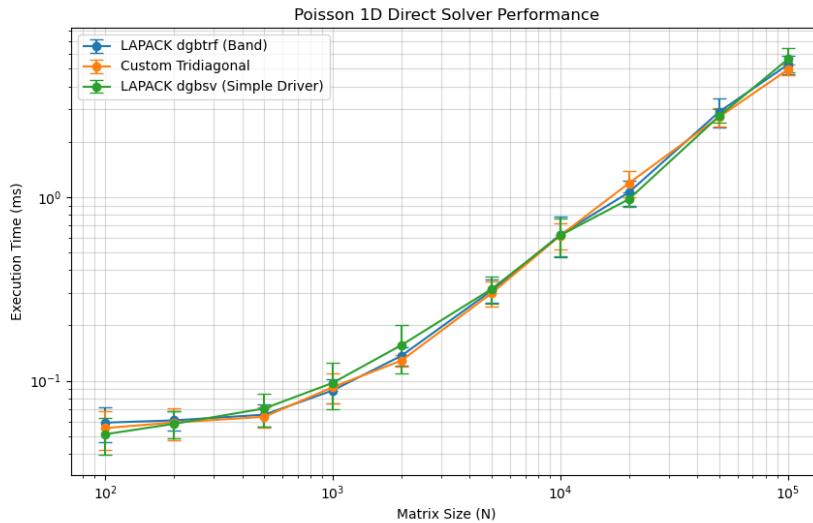


FIGURE 1 – Performance des méthodes directes (échelle Log-Log)

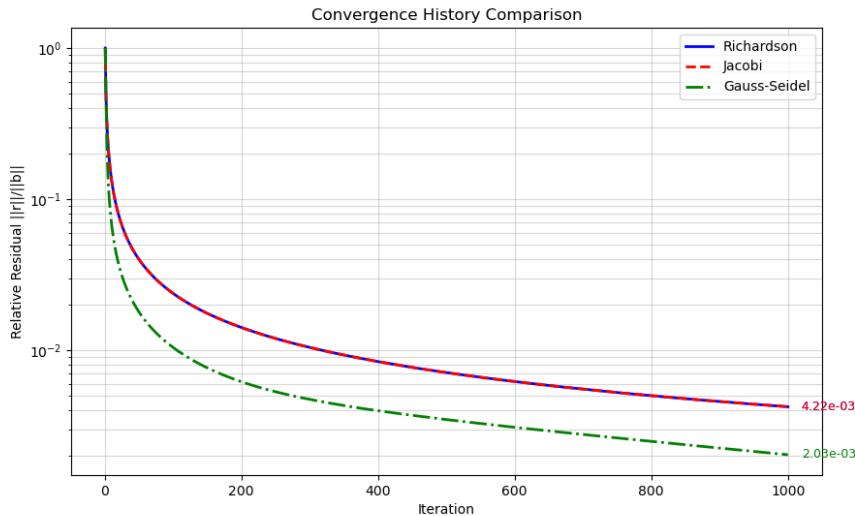
#### 5.4 Convergence des Méthodes Itératives

Méthode	N=10	N=100
Richardson	88	> 1000
Jacobi	88	> 1000
Gauss-Seidel	88	> 1000

TABLE 3 – Nombre d’itérations pour atteindre  $\varepsilon = 10^{-3}$ 

La Figure 2 illustre l’historique de convergence pour  $N = 100$ . On observe que :

- Les courbes de Richardson et Jacobi sont confondues, car Richardson optimal équivaut à Jacobi pour ce problème.
- Gauss-Seidel converge environ deux fois plus vite (pente plus raide), atteignant un résidu plus faible pour le même nombre d’itérations.

FIGURE 2 – Comparaison de l'historique de convergence ( $N = 100$ , 1000 itérations)

## 6 Formats de stockage creux (CSR/CSC)

(Exercice 10)

Dans cette section, nous étendons notre bibliothèque pour supporter les formats de matrices creuses CSR et CSC. Ces formats sont essentiels pour traiter des problèmes multidimensionnels.

*Note : Pour la clarté de l'exposition, les algorithmes ci-dessous sont présentés en pseudo-code de type Matlab, bien que l'implémentation réelle soit en C.*

### 6.1 Implémentation des Structures

Nous avons défini deux nouvelles structures dans `lib_poisson1D.h` :

```
typedef struct {
    double *values; // Valeurs non-nulles
    int *col_ind; // Indices de colonne
    int *row_ptr; // Pointeurs de début de ligne
    int nnz; // Nombre d'éléments non-nuls
    int n; // Dimension
} CSRMatrix;
```

La structure CSCMatrix est analogue, avec `row_ind` et `col_ptr`. Pour la matrice de Poisson 1D, le nombre d'éléments non-nuls est exactement  $3N - 2$ .

### 6.2 Produit Matrice-Vecteur Creux (SpMV)

L'opération critique pour les méthodes itératives est le produit matrice-vecteur  $y = Ax$ .

#### 6.2.1 Format CSR (dcsrsv)

L'algorithme parcourt les lignes ( $i$ ) et les éléments non-nuls ( $k$ ) :

```
for i = 1:n
    s = 0;
    for k = row_ptr(i) : row_ptr(i+1)-1
        s = s + values(k) * x(col_ind(k));
    end
    y(i) = s;
```

```
end
```

### 6.2.2 Format CSC (dcscmv)

L'algorithme parcourt les colonnes ( $j$ ) et accumule dans le vecteur résultat :

```
y = zeros(n,1);
for j = 1:n
    for k = col_ptr(j) : col_ptr(j+1)-1
        y(row_ind(k)) = y(row_ind(k)) + values(k) * x(j);
    end
end
```

## 6.3 Adaptation des Algorithmes Itératifs

Nous avons adapté la méthode de **Richardson** pour utiliser ces nouveaux formats. La logique reste identique à la version bande, seule l'appel au produit matrice-vecteur change :

- IMPLEM=3 : Richardson avec CSR (utilise dcsmrv)
- IMPLEM=4 : Richardson avec CSC (utilise dcscmv)

## 6.4 Validation

Les tests effectués avec  $N = 10$  montrent une convergence correcte :

- Erreur relative obtenue :  $\approx 3.8 \times 10^{-3}$
- Cette erreur est identique à celle obtenue avec le stockage bande.
- Elle correspond à l'erreur de discrétisation du schéma différences finies ( $O(h^2)$  avec  $h \approx 0.1$ ).  
Cela valide l'exactitude de nos routines de construction de matrice et de multiplication matrice-vecteur.

## 7 Conclusion

Ce TP a permis d'implémenter et de comparer différentes méthodes de résolution pour l'équation de la chaleur 1D. Nous avons couvert :

- Les méthodes directes (LU) via LAPACK et une implémentation manuelle optimisée pour tridiagonale.
- Les méthodes itératives classiques (Richardson, Jacobi, Gauss-Seidel).
- L'extension vers les formats de stockage creux (CSR/CSC).

Si le format bande (General Band) reste le plus performant pour ce problème spécifique 1D (structure tridiagonale stricte), l'implémentation des formats CSR et CSC nous a permis de développer des outils génériques capables de traiter des problèmes physiques plus complexes (2D, 3D, maillages non structurés) où la structure de la matrice n'est plus aussi régulière.