

Rapport du TP Calcul numérique Méthodes itératives de base

Bowen LIU

bowen.liu@ens.uvsq.fr

1. Introduction

L'objectif de ce TD/TP est d'appliquer un ensemble d'algorithmes étudiés en cours et TD pour la résolution d'un système linéaire obtenu par discrétisation par la méthode des différences finies de l'équation de la chaleur 1D stationnaire. Les implémentations seront faites en C avec BLAS et LAPACK. Afin de valider on peut nous reposer sur les implémentations Scilab réalisées en TD. Une analyse critique de résultats est demandée à partir de l'étude des complexités en temps et en espace. Pour chaque algorithme, le temps d'exécution en fonction de la taille de la matrice devra être mesuré et des courbes de performances devront être présentées.

2. Travail préliminaire : Etablissement d'un cas de test

2.1 Exercice 2

Pour un calcul efficace, installer d'abord les bibliothèques BLAS et LAPACK sur le système Ubuntu. Configurer ensuite le fichier hôte (.mk) et le document makefile pour tester l'environnement de travail. Après l'exécution, on obtient les résultats suivants :

```
bin/tp_testenv
```

```
----- Test environment of execution for Practical exercises of Numerical  
Algorithmics -----
```

```
The exponential value is e = 2.718282
```

```
The maximum single precision value from values.h is maxfloat = 3.402823e+38
```

```
The maximum single precision value from float.h is flt_max = 3.402823e+38
```

```
The maximum double precision value from float.h is dbl_max = 1.797693e+308
```

```
The epsilon in single precision value from float.h is flt_epsilon = 1.192093e-07
```

The epsilon in double precision value from float.h is `dbl_epsilon = 2.220446e-16`

Test of ATLAS (BLAS/LAPACK) environment

`x[0] = 1.000000, y[0] = 6.000000`

`x[1] = 2.000000, y[1] = 7.000000`

`x[2] = 3.000000, y[2] = 8.000000`

`x[3] = 4.000000, y[3] = 9.000000`

`x[4] = 5.000000, y[4] = 10.000000`

Test DCOPY `y <- x`

`y[0] = 1.000000`

`y[1] = 2.000000`

`y[2] = 3.000000`

`y[3] = 4.000000`

`y[4] = 5.000000`

----- End -----

Le programme de test s'exécute avec succès, ce qui prouve que notre environnement de travail est normal et que nous pouvons continuer à terminer les prochains travaux dans cet environnement de travail.

3. Méthode directe et stockage bande

3.1 Exercice 3

Q1. En langage C, un pointeur vers un type double et la fonction `malloc` sont utilisés pour allouer dynamiquement de la mémoire à la matrice.

Q2. `LAPACK_COL_MAJOR` est une constante de LAPACK, utilisée pour préciser que l'ordre de stockage de la matrice est colonne majeure. Il s'agit de la méthode de stockage par défaut de LAPACK.

Q3. `ld` fait référence à la dimension principale de la matrice, spécifiant la taille de chaque colonne dans la matrice de stockage.

Q4. `dgbmv` est la fonction dans BLAS pour la multiplication matrice-vecteur à une matrice bandes.

Q5. `dgbtrf` effectue une décomposition LU sur une matrice en bandes et renvoie le résultat stocké dans la matrice d'entrée.

Q6. `dgbtrs` utilise les résultats de `dgbtrf` pour résoudre le système linéaire $AX=B$.

Q7. `dgbsv` décompose directement la matrice LU et résout le système $AX=B$.

Q8. Utiliser d'abord `dgbmv` pour calculer la multiplication matrice-vecteur, puis utiliser la soustraction matricielle pour calculer le résidu, puis appeler `dnrm2` pour calculer la norme.

3.2 Exercice 4

Q1. Dans le fichier `lib_poisson1D.c`, j'ai complété la définition de la fonction `set_GB_operator_colMajor_poisson1D`, qui permet de stocker la matrice Poisson1D au format GB colonne majeure.

Q2. Dans `tp_poisson1D_direct.c`, lorsque `set_GB_operator_colMajor_poisson1D` est appelé, nous appelons `cblas_dgbmv` dans la bibliothèque BLAS pour calculer la multiplication matricielle et vectorielle et écrire le résultat dans le fichier `MatrixVectorResult.dat`.

Q3. Nous pouvons utiliser la fonction `relative_forward_error` qui a été définie dans `lib_poisson1D.c` pour calculer l'erreur entre la solution numérique et la solution analytique. Si l'erreur est inférieure à une valeur spécifique (ici je définis la valeur comme $1e-10$), alors la vérification est réussie.

3.3 Exercice 5

Q1. Nous pouvons effectuer une décomposition LU et résoudre ce système linéaire directement via `dgbsv` dans `tp_poisson1D_direct.c`.

Q2.

Afin d'évaluer les performances des méthodes que nous utilisons dans LAPACK,

nous devons mesurer le temps d'exécution du programme et analyser la complexité temporelle.

Pour la décomposition LU (dgbtrf), la complexité temporelle est $O(n \cdot kl \cdot ku)$, où kl et ku sont respectivement la bande inférieure et la bande supérieure. Parce que la matrice tridiagonale $kl=ku=1$, la complexité temporelle peut être simplifiée en $O(n)$

Pour résoudre des systèmes triangulaires (dgbtrs), la complexité temporelle est similaire à la décomposition LU, qui est $O(n \cdot kl \cdot ku)$, et peut être simplifiée en $O(n)$ pour les matrices tridiagonales.

En résumé, on peut penser que la complexité temporelle dépend de la taille de la matrice et de la largeur de la bande.

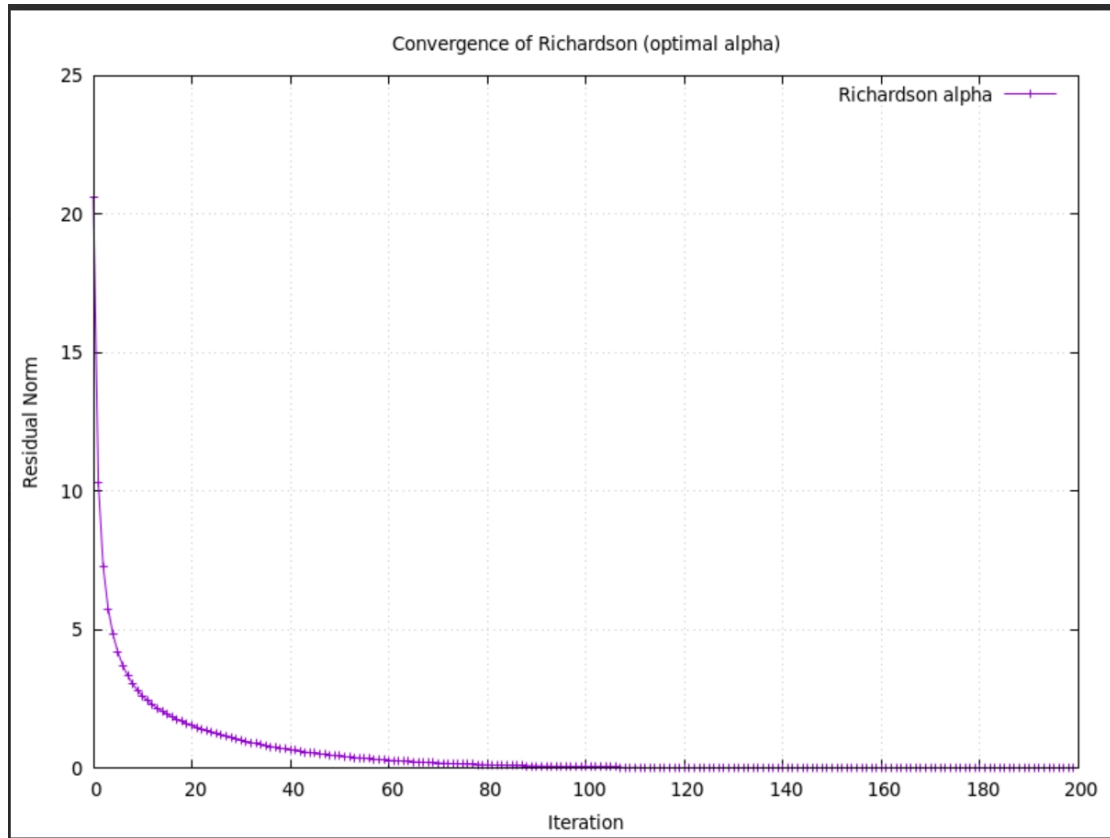
Une autre base importante pour évaluer les performances est de mesurer le temps d'exécution du programme. En chronométrant l'exécution de l'appel de la fonction LAPACK, nous pouvons observer qu'il existe des différences de temps évidentes selon le type d'implémentation (IMPLEM), la taille de la matrice et aussi leur structure de bande.

3.4 Exercise 6

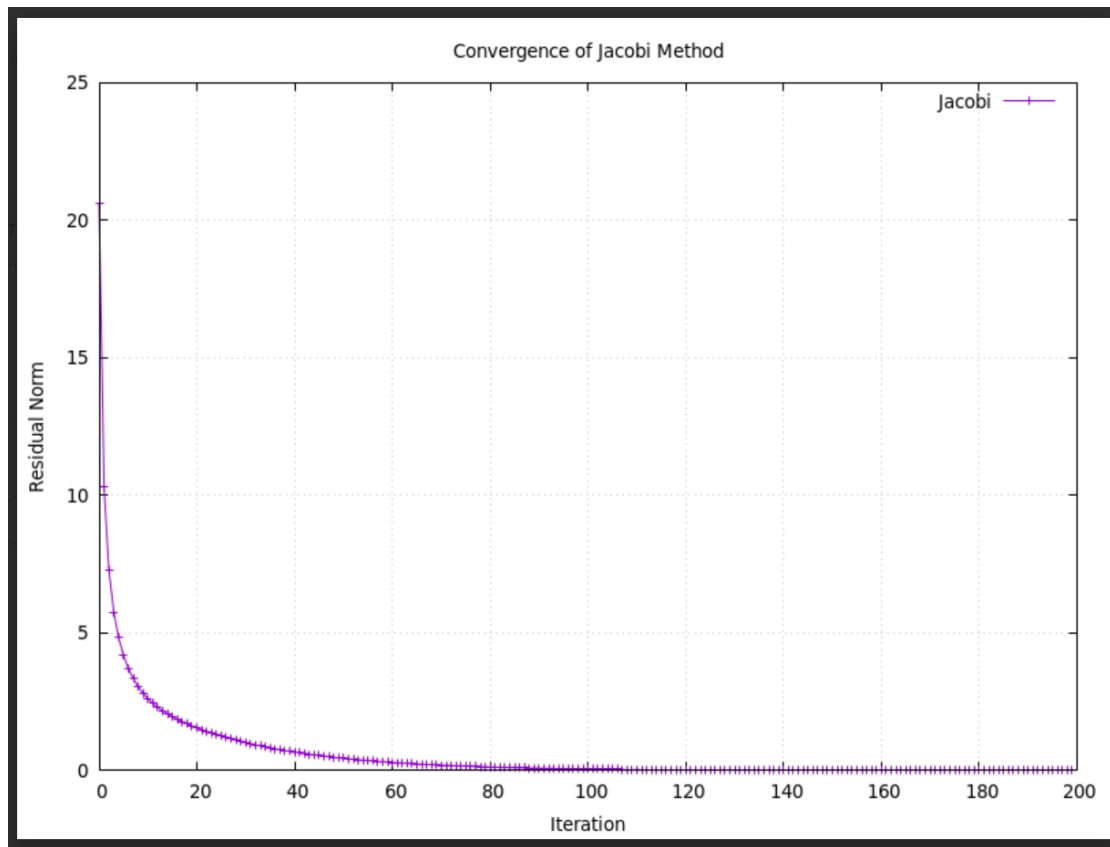
Q1. Dans le fichier lib_poisson1D.c, je définis la fonction dgbtrftridiag pour compléter la décomposition matricielle tridiagonale LU au format GB.

4. Méthode de résolution itérative

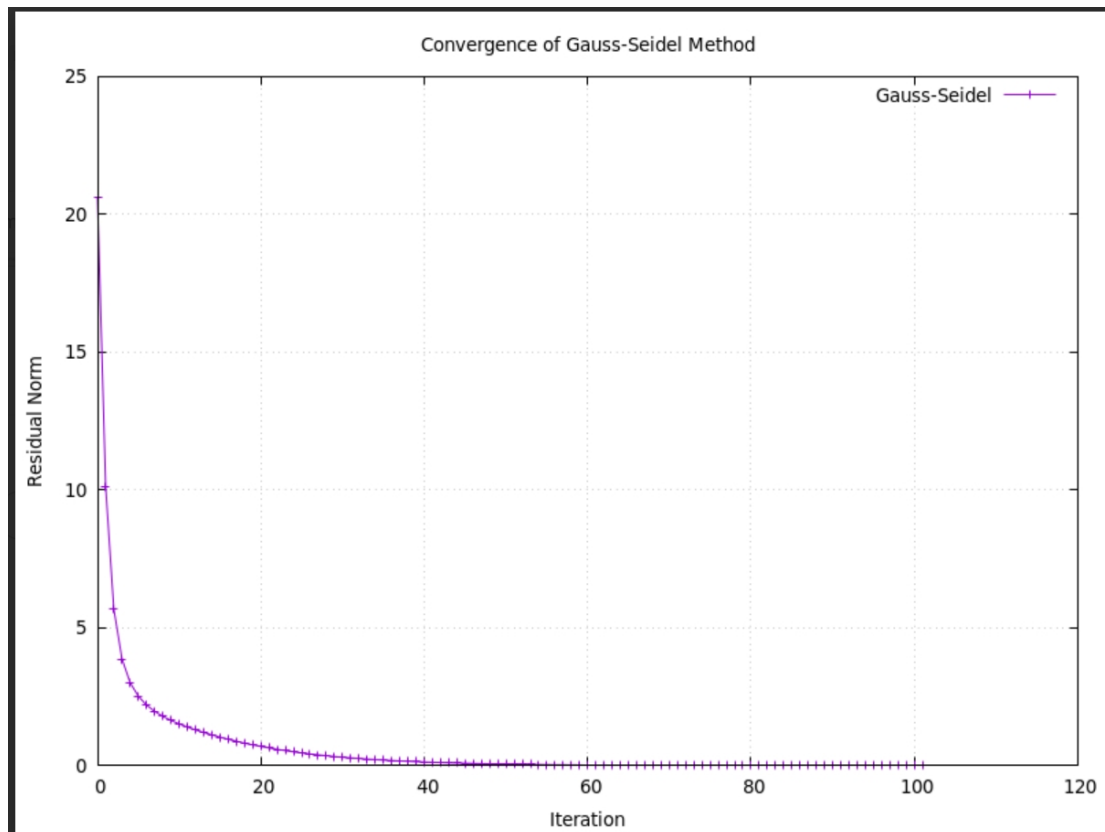
Nous utilisons principalement trois algorithmes itératifs pour résoudre le problème 1DPoisson, en utilisant l'algorithme standard de Richardson, la méthode Jacobi et la méthode Gauss-Seidel pour calculer les résidus et les erreurs par rapport à la solution analytique, et effectuons une analyse de convergence en traçant l'historique de convergence. Dans le programme principal (tp_poisson1D_iter.c), nous exécutons respectivement ces trois méthodes et enregistrons les enregistrements de convergence dans les fichiers RESVEC.dat correspondants et dessinons pour observer la situation de convergence. Les conditions de convergence des trois méthodes d'itération sont les suivantes :



Richardson_Alpha



Jacobi



Gauss-Seidel

5. Autres formats de stockage

Du code est ajouté à la fin du fichier `lib_poisson1D_richardson.c` pour implémenter les formats de stockage CSR et CSC et les fonctions clés `dcsrmv` et `dscmv`. De plus, l'algorithme des opérations matricielles au format CSR ou CSC n'est pas encore apparu dans le code, des recherches supplémentaires sont donc nécessaires pour répondre à ces besoins.

6. Conclusion

Dans ce projet, j'ai écrit du code pour résoudre le problème 1DPoisson, en utilisant la méthode directe et trois méthodes itératives pour le résoudre. Dans ce projet, des fichiers distincts sont utilisés pour chaque fonctionnalité, ce qui facilite la gestion de projet et améliore la lisibilité du code. De plus, j'accorde une attention particulière à l'optimisation de l'algorithme et vérifie son exactitude par comparaison avec la

solution analytique.

Cependant, des améliorations sont encore possibles. J'espère fournir une analyse des performances plus précise et détaillée et mettre en œuvre les méthodes de parallélisation correspondantes pour réduire autant que possible le nombre d'itérations.