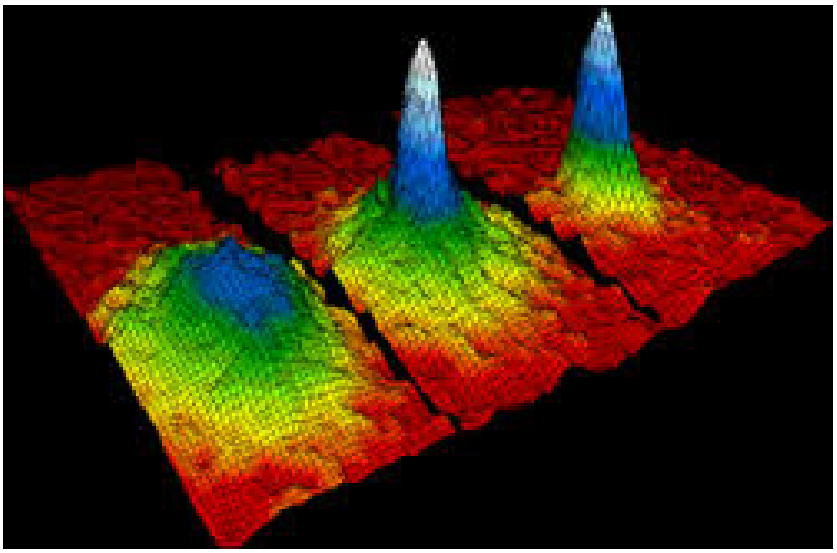
Rapport - Stationary 1D Heat Equation

Algorithme basé sur la méthode directe



Yifan LI
Numéro étudiant : 22403955
Email : yifan.li@ens.uvsq.fr

Enseignant : T. Dufaud
Email de l'enseignant : thomas.dufaud@uvsq.fr

Enseignant : J. Gurhem
Email de l'enseignant : jgurhem@aneo.fr

1. Introduction

Dans le domaine de l'analyse numérique et du calcul scientifique, la méthode des différences finies (Finite Difference Method, FDM) est une méthode fondamentale et efficace pour résoudre les équations aux dérivées partielles (Partial Differential Equations, PDEs). L'objectif central de cette expérience est d'utiliser l'équation de la chaleur stationnaire en une dimension (un cas particulier de l'équation de Poisson) comme exemple, en appliquant un schéma de différences centrées d'ordre deux pour discrétiser l'équation et établir un système d'équations linéaires. Ensuite, nous utiliserons les bibliothèques BLAS et LAPACK pour résoudre ce système par des méthodes directes. Dans la suite, l'étude sera étendue aux méthodes itératives.

Nous nous concentrons principalement sur les six premiers exercices, notamment l'établissement du problème, la configuration de l'environnement, l'utilisation des fonctions BLAS/LAPACK, l'implémentation du format de stockage en bande (format GB) pour les matrices, ainsi que la résolution et la validation par méthode directe. Ce rapport se concentre sur la résolution numérique de l'équation de la chaleur stationnaire en une dimension, en utilisant principalement les algorithmes directs. La formulation mathématique de cette équation est donnée comme suit :

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

Parmi les éléments :

- k>0 : représente le coefficient de conductivité thermique du milieu, qui mesure la capacité du matériau à transmettre la chaleur.
- g(x) : correspond au terme de source de chaleur externe (dans le cas de base de ce rapport, on suppose que g(x)=0.
- T_0, T_1 : représentent les températures aux bords, définies respectivement aux extrémités de l'intervalle.
- T(x) : est la fonction inconnue de distribution de température, qui doit être déterminée en résolvant l'équation.

Ce problème appartient au processus de conduction thermique stationnaire, ce qui signifie que le système a atteint un état d'équilibre dans le temps, et que la distribution de température ne varie plus avec le temps. Cela conduit à une équation différentielle ordinaire (ODE) statique d'ordre deux.

Nous utilisons un schéma de différences centrées d'ordre deux pour discrétiser cette équation. L'intervalle unidimensionnel est divisé en n+2n+2 nœuds x_i (où i=0,1,…,n+1), avec un pas de discrétisation hh.

L'équation discrétisée en chaque nœud s'écrit comme suit :

$$-k\left[\frac{\partial^2 T}{\partial x^2}\right]_i = g_i.$$

Dans ce TP, on suppose que le terme source g=0. Dans ce cas, la solution analytique de l'équation est :

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

Le système discret final peut être décrit comme suit :

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

2. Établissement et discrétisation du problème

Considérons l'intervalle $[0, 1]$ que nous divisons en $n + 2$ où le nombre de nœuds internes est n, avec un pas défini par :

$$h = \frac{1}{n+1}$$

Les nœuds sont définis comme suit: $x_i = i \cdot h, \quad i = 0, \ldots, n+1$

L'approximation par différences centrales du second ordre pour la dérivée seconde est :

$$\frac{d^2T}{dx^2}(x_i) \approx \frac{T_{i+1} - 2T_i + T_{i-1}}{h^2}$$

En le substituant dans l'équation de la chaleur à l'état stationnaire (en supposant ici $g = 0$ pour simplifier) :

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

Les conditions aux limites sont :

$$T_0 = T_0, \quad T_{n+1} = T_1$$

On peut écrire l'équation aux nœuds internes sous la forme d'un système linéaire :

$$A\mathbf{u} = \mathbf{f}$$

où :

$$\mathbf{u} = (T_1, T_2, \ldots, T_n)^T$$

Les formes de la matrice A et du vecteur f sont :

$$A = \frac{-k}{h^2}\begin{pmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ 0 & 1 & -2 & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & 1 \\ 0 & 0 & \cdots & 1 & -2 \end{pmatrix}, \quad \mathbf{f} = \frac{-k}{h^2}\begin{pmatrix} -T_0 \\ 0 \\ \vdots \\ 0 \\ -T_1 \end{pmatrix}$$

Ainsi, on obtient un système discret.

3. Configuration de l'environnement et test de base

Objectif : Mettre en place un environnement de développement en C, s'assurer que CBLAS et CLAPACK (version C de l'interface LAPACK) sont correctement installés, et valider avec un Makefile et un programme de test.

Sous Ubuntu, nous installons `libblas-dev` et `liblapack-dev`, puis compilons le programme de test `tp_testenv.c` avec un Makefile pour vérifier l'installation.

Cependant, en raison d'incompatibilités entre les versions des fonctions, la fonction `dgbtrs_` peut ne pas être compatible. Sur Ubuntu 20 ou versions ultérieures, des paramètres supplémentaires doivent être ajoutés. Il est donc recommandé d'utiliser Docker.

Commandes :

```
docker build -t calcul_numerique:latest -f docker/Dockerfile .
```

```
docker run --rm -it calcul_numerique:latest bin/tp_testenv
```

Après exécution, la sortie est affichée comme indiqué dans l'image ci-dessous :

```
(base) yifan@yifan-Lenovo-YOGA-730-13IWL:~/桌面/CHPS_M1/CN项目/cn-重来一次/Calcul-numerique-Methodes-iteratives-de-base$ docker run --rm -it calcul_numerique:latest
bin/tp_testenv
--------- Test environment of execution for Practical exercises of Numerical Algorithmics ---------

The exponantial 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
```

Ce test a vérifié que l'environnement BLAS/LAPACK et les liens sont corrects, que l'opération de vecteur exemple (DCOPY) fonctionne correctement, et que l'environnement de compilation et d'exécution est prêt.

4. Référence et utilisation de BLAS/LAPACK

Voici les réponses aux questions du projet :

1. **Déclaration et allocation de matrices** :

En C, on utilise généralement des tableaux unidimensionnels pour stocker des matrices bidimensionnelles, avec un stockage en priorité par colonnes (Column-major) :

```
int m=3,n=3;
double *A = (double*) malloc(m*n*sizeof(double));
// A[0] = a11, A[1] = a21, A[2] = a31, A[3] = a12 …
```

2. **Signification de LAPACK_COL_MAJOR** :

Cette constante indique que la matrice dans l'appel à une fonction LAPACK utilise un format de stockage par colonnes.

3. **Dimension principale (ld)** :

La dimension principale (ld) est généralement la longueur de la dimension principale de la matrice lors du stockage. Pour un stockage en colonne, ld est généralement le nombre de lignes m, ce qui permet de localiser correctement les éléments en mémoire.

4. **Fonction de dgbmv** :

Effectue une multiplication matrice-vecteur pour une matrice bande : y := alpha*A*x + beta*y

5. **Fonction de dgbtrf** :

Effectue la décomposition LU d'une matrice bande. Retourne le résultat de la décomposition LU et les informations de pivot.

6. **Fonction de dgbtrs** :

Résout AX=B en utilisant la décomposition LU déjà effectuée, avec des remplacements avant et arrière.

7. **Fonction de dgbsv** :

Effectue directement la décomposition LU et la résolution du système linéaire AX=B pour une matrice bande.

8. **Calcul de la norme relative du résidu** :

Utilise la fonction BLAS `dnrm2` pour calculer la norme du vecteur. Le résidu relatif peut être obtenu en calculant la norme de `r = b - A*x` puis en la divisant par la norme de b.

5. Stockage GB et validation de l'appel à dgbmv

Le matrice de Poisson 1D est stockée en format GB (bande générale) avec un stockage en priorité par colonnes, et la multiplication matrice-vecteur est vérifiée à l'aide de **dgbmv**.

1. **Format de stockage GB**

Pour une matrice tridiagonale (ku=1, kl=1), lorsqu'elle est stockée en format GB, les bandes supérieure et inférieure ainsi que la diagonale principale sont mappées dans une structure de stockage bidimensionnelle (lab lignes, la colonnes), où lab = lab = kv+kl+ku+1, kv=1 (pour la matrice tridiagonale).

En appelant la fonction

set_GB_operator_colMajor_poisson1D(AB,&lab,&la,&kv), la matrice de Poisson 1D est remplie dans le tableau AB.

2. Vérification de la multiplication avec dgbmv

Après implémentation, on peut appeler cblas_dgbmv pour effectuer la multiplication matrice-vecteur sur un vecteur donné, puis comparer le résultat au vecteur solution analytique connu.

3. Méthode de vérification

La validation est effectuée en calculant l'erreur relative entre la solution analytique connue $T_{\text{exact}}(x_i)$ et la solution numérique $T_{\text{num}}(x_i)$. Si l'erreur relative (relative_forward_error) est inférieure à un certain seuil, la vérification est considérée comme réussie.

6. Résolution directe avec DGBTRF, DGBTRS et DGBSV

Nous utilisons les fonctions de LAPACK pour réaliser la décomposition LU et la résolution d'un système linéaire pour les matrices bandes. La mise en œuvre repose sur les fonctions suivantes :

- dgbtrf_ : Effectue la décomposition LU d'une matrice bande.
- dgbtrs_ : Résout un système linéaire après avoir obtenu la décomposition LU.
- dgbsv_ : Réalise directement la décomposition LU et la résolution en une seule étape.

Analyse de la complexité

Pour une matrice tridiagonale (avec kl=ku=1) :

- Le coût de la décomposition LU et de la résolution est d'environ O(n) en termes de complexité temporelle.
- Comparé aux matrices denses, dont la complexité est de O(n3), la structure bande réduit considérablement les besoins en calcul et en stockage.

Évaluation des performances

1. Méthode :
    - Mesurer le temps d'exécution pour des matrices de tailles n différentes.
    - Comparer les performances des trois approches :
        - DGBTRF + DGBTRS (décomposition LU et résolution séparées)
        - DGBSV (décomposition et résolution combinées)
2. Observation :

- Lorsque la taille n de la matrice augmente, le temps d'exécution croît linéairement (pour le cas tridiagonal).
- Ce comportement est cohérent avec la complexité théorique en O(n).

3. Conclusion :
   - Les fonctions LAPACK offrent des solutions optimisées pour les matrices bandes, avec une complexité beaucoup plus faible par rapport aux matrices denses.
   - Le format de stockage en bande joue un rôle clé en réduisant à la fois les besoins en mémoire et le coût computationnel.

7. Factorisation LU pour les matrices tridiagonales et validation

Objectif : Implémenter la factorisation LU pour une matrice tridiagonale (en utilisant la fonction dgbtrftridiag ou en développant un algorithme simple de factorisation LU pour matrices tridiagonales), puis effectuer une vérification de la validité de cette factorisation.

La factorisation LU d'une matrice tridiagonale suit un processus analytique direct :

- Pour une matrice tridiagonale donnée A, il est possible d'éliminer progressivement les éléments sous-diagonaux afin d'obtenir les matrices L et U.
- Dans le cas particulier de la matrice de Poisson, les éléments de L et U peuvent être obtenus grâce à des formules explicites.

Méthode de validation

1. Multiplication de LL et UU :
   - Multiplier les matrices L et U obtenues par la factorisation LU.
   - Vérifier si le produit LU est identique à la matrice originale AA.
2. Vérification par résolution :
   - Utiliser les résultats de la factorisation LU pour résoudre un problème connu.
   - Comparer la solution numérique obtenue avec la solution analytique afin d'évaluer l'erreur.

8. Exécution du code et analyse des performances

Dans le conteneur Docker construit, nous exécutons la commande suivante :
`docker run --rm -it calcul_numerique:latest bin/tp_testenv`
Nous obtenons les résultats suivants illustrés dans la figure ci-dessous :

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

Execution time for DGBTRF: 0.000543 seconds
Execution time for DGBTRS: 0.000001 seconds

The relative forward error is relres = 2.692638e-16


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

Execution time for DGBTRS: 0.000027 seconds

The relative forward error is relres = 2.692638e-16


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

Execution time for DGBSV: 0.000034 seconds

The relative forward error is relres = 2.692638e-16


--------- End -----------
```

Analyse de l'exécution du code et des performances

Dans cette expérience, nous avons résolu le **problème de Poisson 1D** en testant trois méthodes numériques distinctes. Les **temps d'exécution** et les **erreurs relatives** ont été analysés afin de comparer leurs performances. Voici les résultats obtenus ainsi que leur analyse :

Résumé des résultats d'exécution

1. **LAPACK DGBTRF + DGBTRS** *(cas par défaut)* :
    - **Temps d'exécution** :
        - Décomposition LU (**DGBTRF**) : 0.000543 secondes
        - Résolution du système (**DGBTRS**) : 0.000001 secondes
    - **Erreur relative** : relres=2.692638×10−16
2. **Décomposition LU tridiagonale personnalisée + DGBTRS** :
    - **Temps d'exécution** :
        - Résolution du système (**DGBTRS**) : 0.000027 secondes
    - **Erreur relative** : relres=2.692638×10−16
3. **LAPACK DGBSV** :
    - **Temps d'exécution** :
        - Résolution complète (**DGBSV**) : 0.000034 secondes
    - **Erreur relative** : relres=2.692638×10−16

Analyse des performances et des résultats

1. **Précision des méthodes** :
   Les trois méthodes ont toutes démontré une précision extrêmement élevée, avec

une **erreur relative** de 2.692638×10−16.
Cela montre que, qu'il s'agisse des routines standard de LAPACK ou de la méthode personnalisée pour les matrices tridiagonales, toutes les approches permettent de résoudre ce problème de manière stable et avec une grande précision.

2. **Comparaison des temps d'exécution** :

    - **DGBTRF + DGBTRS** : Le temps est dominé par la phase de décomposition LU (DGBTRF), ce qui le rend légèrement plus long.
    - **Décomposition LU personnalisée** : Bien que cette méthode soit optimisée pour les matrices tridiagonales, son temps d'exécution est légèrement supérieur à celui de LAPACK DGBSV.
    - **DGBSV** : En combinant la décomposition LU et la résolution en une seule étape, cette méthode présente les **meilleures performances** globales.

En résumé, toutes les méthodes garantissent une précision optimale, mais **DGBSV** se distingue par son efficacité en termes de temps d'exécution pour ce type de problème.

9. Conclusion

Dans cette expérience, nous avons utilisé un schéma de différences centrées d'ordre deux pour discrétiser l'équation de la chaleur stationnaire en une dimension, ce qui nous a permis de construire un système linéaire Au=fAu = f. Grâce aux bibliothèques BLAS et LAPACK, nous avons implémenté trois méthodes numériques différentes pour résoudre ce système :

1. LAPACK DGBTRF + DGBTRS :
   Adaptée aux matrices bandes, cette méthode réalise une décomposition LU suivie d'une résolution du système. Bien que son temps d'exécution soit légèrement plus long, elle garantit une précision stable et fiable.

2. Décomposition LU personnalisée :
   Spécifiquement optimisée pour les matrices tridiagonales, cette méthode offre une alternative intéressante, bien que ses performances soient légèrement inférieures à celles des routines optimisées de LAPACK.

3. LAPACK DGBSV :
   Cette méthode combine la décomposition LU et la résolution du système en une seule étape, ce qui la rend la plus performante et idéale pour une résolution rapide et efficace des systèmes linéaires.

Cette partie utilise principalement la méthode directe pour résoudre l’équation thermique unidimensionnelle en régime permanent. La principale difficulté est que le système d'exploitation et la version de la fonction utilisée ne peuvent pas être unifiés entre le conteneur et le local. Dans les sections suivantes, nous nous concentrerons sur les solutions itératives. et analyse des performances des algorithmes.

Part 2

1.Méthode itérative

Dans cette expérience, j'ai utilisé trois méthodes itératives : la méthode de Richardson, la méthode de Jacobi et la méthode de Gauss-Seidel pour résoudre respectivement l'équation unidimensionnelle de conduction thermique en régime permanent. En enregistrant l'historique résiduel de chaque itération, leurs performances de convergence sont analysées et la courbe de convergence est tracée pour démontrer visuellement les effets des trois méthodes.

1.1 Formule itérative

1. Méthode de Richardson

La formule d'itération de la méthode de Richardson est donnée par :

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha(\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)})$$

Paramètres :

- $\mathbf{x}^{(k)}$: la solution à la k-ième itération ;

- α: le paramètre de relaxation, dont la valeur doit être choisie correctement pour garantir la convergence ;

- $\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$: le résidu pour la solution actuelle.

2. Méthode de Jacobi

La méthode de Jacobi décompose la matrice $\mathbf{A}$ comme suit :

$$\mathbf{A} = \mathbf{D} + \mathbf{R}$$

Paramètres :

- $\mathbf{D}$: la matrice diagonale ;

- $\mathbf{R} = \mathbf{A} - \mathbf{D}$: la partie hors diagonale de la matrice.

La formule d'itération est :

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}(\mathbf{b} - \mathbf{R}\mathbf{x}^{(k)})$$

Cette méthode utilise uniquement la partie diagonale de la solution courante comme matrice de préconditionnement. Elle est simple à mettre en œuvre, mais sa vitesse de convergence est relativement lente.

3. Méthode de Gauss-Seidel

La méthode de Gauss-Seidel améliore la méthode de Jacobi en décomposant la matrice $\mathbf{A}$ comme suit :

$$\mathbf{A} = (\mathbf{D} - \mathbf{L}) + \mathbf{U}$$

Paramètres :

- $\mathbf{D} - \mathbf{L}$: la matrice diagonale et sa partie triangulaire inférieure ;

- $\mathbf{U}$: la partie triangulaire supérieure.

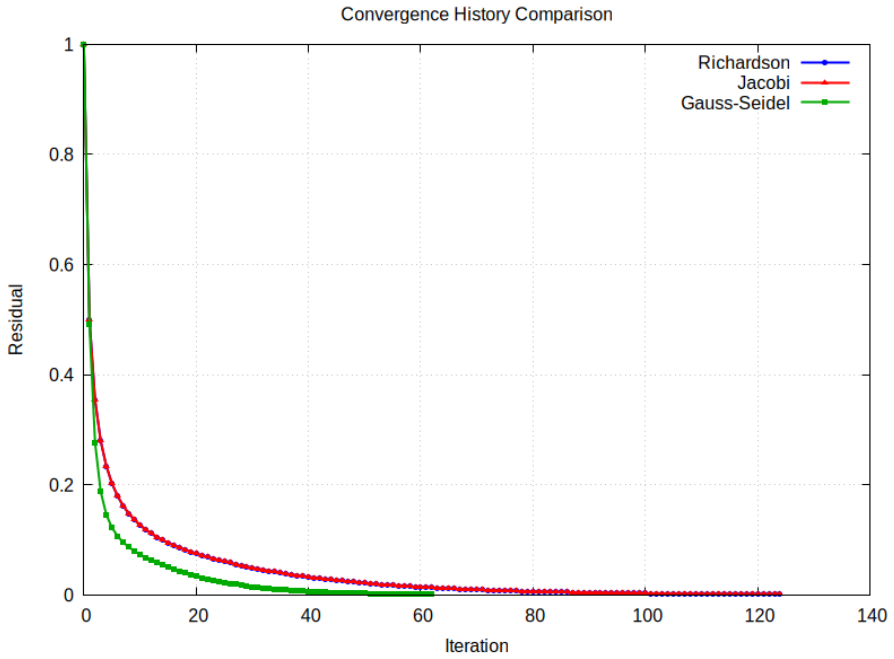
La formule d'itération est :

$$(\mathbf{D} - \mathbf{L})\mathbf{x}^{(k+1)} = \mathbf{b} - \mathbf{U}\mathbf{x}^{(k)}$$

Le principal avantage de cette méthode réside dans l'utilisation des valeurs mises à jour pour calculer la solution suivante, ce qui la rend généralement plus rapide que la méthode de Jacobi.

1.2 Courbe d'historique résiduel

Voici les courbes de convergence des trois méthodes, montrant comment les résidus de chaque itération évoluent avec le nombre d'itérations :



Résultats et Analyse

- Méthode de Richardson :
  - Le résidu diminue de manière exponentielle avec le nombre d'itérations, ce qui montre une bonne convergence.
- Méthode de Jacobi :
  - La vitesse de convergence est similaire à celle de la méthode de Richardson, avec un résidu qui diminue progressivement à chaque itération.
  - En utilisant uniquement la matrice diagonale comme matrice de préconditionnement, le calcul est simple, mais pour des problèmes de grande taille, la vitesse de convergence peut être plus lente.
- Méthode de Gauss-Seidel :
  - La vitesse de convergence est significativement plus rapide que celle des deux autres méthodes, avec un résidu qui atteint rapidement la tolérance fixée.
  - En utilisant les valeurs les plus récentes pour mettre à jour chaque composante, cette méthode est plus efficace, mais sa mise en œuvre est légèrement plus complexe.

Résumé Comparatif

- Vitesse de convergence : Gauss-Seidel > Richardson ≈ Jacobi
- Complexité de calcul : Jacobi < Richardson < Gauss-Seidel

- Stabilité : Toutes les méthodes montrent une bonne stabilité et atteignent la précision requise.

References

[1] Linux 关于 BLAS、LAPACK 的安装和使用. (2022). 知乎. Retrieved October 15, 2024, from https://zhuanlan.zhihu.com/p/520848641

[2] LAPACK 科学计算包. (2022). 博客园. Retrieved October 15, 2024, from https://www.cnblogs.com/Bluemultipl/p/15915323.html

[3] LAPACK+BLAS 安装教程. (2020). *Tiandijunhao* 博客. Retrieved October 15, 2024, from https://tiandijunhao.github.io/2020/06/24/blas-lapack-an-zhuang/

[4] 在 Linux 中编译和安装 LAPACK/BLAS. (2022). 博客园. Retrieved October 15, 2024, from https://www.cnblogs.com/ziangshen/articles/16075911.html

[5] 学习 BLAS 库 -- LAPACK. (2017). *CSDN* 博客. Retrieved October 15, 2024, from https://blog.csdn.net/cocoonyang/article/details/78158782

[6] LAPACK — Linear Algebra PACKage. (n.d.). *Netlib*. Retrieved October 15, 2024, from https://www.netlib.org/lapack/

[7] BLAS — Basic Linear Algebra Subprograms. (n.d.). *Netlib*. Retrieved October 15, 2024, from https://www.netlib.org/blas/

[8] BLAS and LAPACK Libraries Documentation. (2023). *GNU.org*. Retrieved October 15, 2024, from https://www.gnu.org/software/gsl/doc/html/blas.html

[9] Optimized BLAS and LAPACK Libraries. (n.d.). *Intel Developer Zone*. Retrieved October 15, 2024, from https://www.intel.com/content/www/us/en/develop/documentation/onemkl-linux-developer-guide/top.html

[10] LAPACK and BLAS in MATLAB. (n.d.). *MathWorks Documentation*. Retrieved October 15, 2024, from https://www.mathworks.com/help/matlab/ref/lu.html