REPUBLIQUE DU BENIN

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE (MESRS)

UNIVERSITÉ NATIONALE DES SCIENCES, TECHNOLOGIES, INGÉNIERIE ET MATHÉMATIQUES (UNSTIM)

ÉCOLE NATIONALE SUPÉRIEURE DES GÉNIE MATHÉMATIQUE ET MODÉLISATION (ENSGMM-ABOMEY)

UTILSATION DE INTEL MKL EN CALCUL SCIENTIFIQUE : BLAS ET LAPACK

Membres du groupe :

AHOTONHOUN Aimé Césaire

BONOU Justus

HANDJEMEDJI Ezéchiel

PROJET DE CALCUL SCIENTIFIQUE

1^{re} année - CYCLE D'INGÉNIEUR

Sous la supervision de : Dr. Ing. Carlos AGOSSOU

ANNÉE ACADÉMIQUE : 2025-2026

Table des matières

1 Introduction à Intel Math Kernel Library (MKL)

1.1 Présentation générale

Intel Math Kernel Library (MKL) est une bibliothèque mathématique optimisée pour les processeurs Intel. Elle fournit des implémentations haute performance des routines BLAS (Basic Linear Algebra Subprograms), LAPACK (Linear Algebra PACKage), ScaLAPACK, ainsi que des solveurs pour les équations différentielles, les transformées de Fourier et les problèmes d'optimisation. MKL est particulièrement adaptée aux calculs scientifiques intensifs grâce à son optimisation pour l'architecture x86-64 et son support du parallélisme multithread via OpenMP et TBB.

1.2 Applications pratiques de BLAS et LAPACK dans la vie active

Les bibliothèques BLAS et LAPACK, via Intel MKL, sont omniprésentes dans de nombreux domaines professionnels et industriels :

1.2.1 Ingénierie et Simulation Numérique

Dans le domaine de l'ingénierie aéronautique, par exemple, les simulations de dynamique des fluides computationnelle (CFD) reposent massivement sur la résolution de systèmes linéaires de grande taille. Les équations de Navier-Stokes, discrétisées sur des maillages complexes de plusieurs millions de points, génèrent des matrices creuses dont la manipulation efficace nécessite des routines BLAS optimisées. La stabilité et la précision numérique offertes par LAPACK sont cruciales pour garantir la fiabilité des prédictions aérodynamiques.

1.2.2 Finance Quantitative

Le secteur financier utilise intensivement ces bibliothèques pour la modélisation des risques, l'optimisation de portefeuilles, et le pricing d'options exotiques. Les méthodes de Monte-Carlo, les algorithmes de réduction de dimensionnalité, et les calculs de corrélations entre actifs impliquent des opérations matricielles massives. La performance d'Intel MKL permet de réduire significativement les temps de calcul, un avantage compétitif décisif dans les marchés financiers où chaque milliseconde compte.

1.2.3 Recherche Scientifique

En physique des matériaux, la résolution de l'équation de Schrödinger pour des systèmes complexes nécessite le calcul de valeurs propres de matrices de très grande dimension. En imagerie médicale, la reconstruction tomographique repose sur des algorithmes de décomposition en valeurs singulières (SVD) fournis par LAPACK. Dans tous ces domaines, la robustesse numérique et la performance d'Intel MKL sont essentielles.

1.2.4 Intelligence Artificielle et Apprentissage Automatique

Les réseaux de neurones profonds impliquent des millions d'opérations matricielles lors de l'entraînement et de l'inférence. Bien que des bibliothèques spécialisées comme cuDNN existent pour les GPU, de nombreuses étapes de prétraitement et de post-traitement sur CPU utilisent intensivement BLAS et LAPACK. L'analyse en composantes principales (PCA), méthode fondamentale de réduction de dimensionnalité, repose directement sur le calcul de vecteurs propres via LAPACK.

1.3 Installation et configuration

1.3.1 Installation sur Linux

```
# Méthode 1 : Via le gestionnaire de paquets
sudo apt-get install intel-mkl

# Méthode 2 : Téléchargement depuis le site Intel
# 1. Télécharger le package depuis intel.com
# 2. Extraire et exécuter l'installateur
tar -xzvf l_mkl_2024.0.0.tgz
cd l_mkl_2024.0.0
sudo ./install.sh

# Configuration des variables d'environnement
export MKLROOT=/opt/intel/mkl
export LD_LIBRARY_PATH=$MKLROOT/lib/intel64:$LD_LIBRARY_PATH
```

1.3.2 Installation sur Windows

— Télécharger Intel oneAPI Base Toolkit depuis le site Intel
— Exécuter l'installateur graphique
— Configurer les variables d'environnement dans Visual Studio

1.3.3 Liaison avec un programme C/C++

```
# Compilation avec gcc
gcc -o mon_programme mon_programme.c -lmkl_rt -lpthread -lm -ldl

# Avec Intel Compiler
icc -o mon_programme mon_programme.c -mkl
```

2 Problème physique : Vibrations d'une membrane non uniforme

2.1 Contexte physique

On s'intéresse à une membrane carrée de dimension 1×1 mètre, tendue et fixée sur tout son pourtour. Cette membrane peut représenter divers systèmes physiques :
— Un **drum** ou instrument de percussion
— Un **capteur vibratoire** en ingénierie
— Une **plaque instrumentale** en acoustique
La particularité de cette membrane réside dans sa **non-uniformité** :
— La tension varie spatialement sur le domaine
— La densité de masse n'est pas constante
— Un obstacle central modifie localement la rigidité

2.2 Modèle mathématique continu

Le comportement vibratoire est décrit par l'équation aux dérivées partielles suivante :

$$-\nabla \cdot \big(p(x,y)\nabla u(x,y)\big) + q(x,y)u(x,y) = \lambda\, w(x,y)\, u(x,y), \quad (x,y) \in \Omega$$

avec :
— $\Omega = [0,1] \times [0,1]$: domaine carré
— $u(x,y)$: déplacement vertical (mode propre)
— $\lambda = \omega^2$: valeur propre (carré de la fréquence angulaire)
— $p(x,y)$: coefficient de tension variable
— $w(x,y)$: densité de masse variable
— $q(x,y)$: potentiel représentant l'obstacle central

2.3 Conditions aux limites et données

2.3.1 Conditions aux limites

Dirichlet homogène sur tout le bord :

$$u(x,y) = 0 \quad \text{sur } \partial\Omega$$

2.3.2 Fonctions coefficients

$$\begin{aligned} p(x,y) &= 1 + 0.5\sin(2\pi x)\cos(2\pi y) \\ w(x,y) &= 1 + 0.3xy \\ q(x,y) &= 50e^{-50[(x-0.5)^2+(y-0.5)^2]} \end{aligned}$$

3 Discrétisation par différences finies

3.1 Maillage du domaine

On considère un maillage uniforme de $N \times N$ points intérieurs. Le pas spatial est :

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

Les coordonnées des points sont :

$$x_i = i \cdot h, \quad y_j = j \cdot h, \quad i,j = 0,1,\ldots,N+1$$

Les points intérieurs (inconnues) sont pour $i,j = 1,\ldots,N$.

3.2 Discrétisation des opérateurs différentiels

3.2.1 Opérateur Laplacien avec coefficient variable

L'opérateur $\nabla \cdot (p\nabla u)$ est discrétisé en utilisant un schéma aux différences finies centrées :

$$\nabla\cdot(p\nabla u) \approx \frac{1}{h^2}\left[p_{i+\frac{1}{2},j}(u_{i+1,j}-u_{i,j}) - p_{i-\frac{1}{2},j}(u_{i,j}-u_{i-1,j})\right] + \frac{1}{h^2}\left[p_{i,j+\frac{1}{2}}(u_{i,j+1}-u_{i,j}) - p_{i,j-\frac{1}{2}}(u_{i,j}-u_{i,j-1})\right]$$

où $p_{i+\frac{1}{2},j} = \frac{p_{i,j}+p_{i+1,j}}{2}$ et $p_{i,j+\frac{1}{2}} = \frac{p_{i,j}+p_{i,j+1}}{2}$.

3.2.2 Discrétisation complète

Pour chaque point intérieur (i, j), l'équation discrétisée s'écrit :

$$-\frac{1}{h^2}\left[p_{i+\frac{1}{2},j}(u_{i+1,j}-u_{i,j})-p_{i-\frac{1}{2},j}(u_{i,j}-u_{i-1,j})\right] - \frac{1}{h^2}\left[p_{i,j+\frac{1}{2}}(u_{i,j+1}-u_{i,j})-p_{i,j-\frac{1}{2}}(u_{i,j}-u_{i,j-1})\right] + q_{i,j}u_{i,j} = \lambda w_{i,j}u_{i,j} \quad (1)$$

3.3 Formulation matricielle

3.3.1 Numérotation des inconnues

On utilise une numérotation lexicographique (ligne par ligne) :

$$k = (i-1)\times N + (j-1), \quad k = 0, \ldots, N^2-1$$

3.3.2 Construction des matrices

On obtient un problème aux valeurs propres généralisé de la forme :

$$A\mathbf{u} = \lambda B\mathbf{u}$$

où :
— $\mathbf{u} \in \mathbb{R}^{N^2}$: vecteur des déplacements aux nœuds
— $A \in \mathbb{R}^{N^2\times N^2}$: matrice de rigidité (creuse, symétrique)
— $B \in \mathbb{R}^{N^2\times N^2}$: matrice de masse (diagonale)

3.3.3 Structure de la matrice A

Pour un nœud intérieur (i, j), les contributions non nulles sont :

$$\begin{aligned}
A_{k,k} &= \frac{1}{h^2}(p_{i+\frac{1}{2},j}+p_{i-\frac{1}{2},j}+p_{i,j+\frac{1}{2}}+p_{i,j-\frac{1}{2}})+q_{i,j}\\
A_{k,k+1} &= -\frac{1}{h^2}p_{i,j+\frac{1}{2}} \quad (\text{si } j<N)\\
A_{k,k-1} &= -\frac{1}{h^2}p_{i,j-\frac{1}{2}} \quad (\text{si } j>1)\\
A_{k,k+N} &= -\frac{1}{h^2}p_{i+\frac{1}{2},j} \quad (\text{si } i<N)\\
A_{k,k-N} &= -\frac{1}{h^2}p_{i-\frac{1}{2},j} \quad (\text{si } i>1)
\end{aligned}$$

3.3.4 Matrice de masse B

La matrice B est diagonale avec :

$$B_{k,k} = w_{i,j}$$

4 Solveurs Intel MKL pour problèmes aux valeurs propres

4.1 Classification des solveurs disponibles

Intel MKL propose plusieurs familles de solveurs pour les problèmes aux valeurs propres :

Solveur	Type de matrice	Problème	Fonction MKL
`dsyev`	Dense, symétrique	Standard	`LAPACK_dsyev`
`dsyevd`	Dense, symétrique	Standard (diviser-pour-regner)	`LAPACK_dsyevd`
`dsygv`	Dense, symétrique	Généralisé	`LAPACK_dsygv`
`dsygvd`	Dense, symétrique	Généralisé (diviser-pour-regner)	`LAPACK_dsygvd`
FEAST	Creuse, symétrique	Standard/Généralisé	`mkl_sparse_ee`
PARDISO+	Creuse, symétrique	Standard	`pardiso_eigensolver`

TABLE 1 – Solveurs de valeurs propres dans Intel MKL

4.2 Solveur `dsygv` pour matrices denses

4.2.1 Description

Le solveur `dsygv` résout le problème aux valeurs propres généralisé $A\mathbf{x} = \lambda B\mathbf{x}$ pour des matrices symétriques réelles. Il utilise la réduction au problème standard suivie de l'algorithme QR.

4.2.2 Interface en C

```
void dsygv(
    int *itype,      // Type de problème (1: A*x = lambda*B*x)
    char *jobz,      // 'N': valeurs propres seulement, 'V': vecteurs aussi
    char *uplo,      // 'U': partie supérieure stockée, 'L': partie inférieure
    int *n,          // Ordre des matrices
    double *A,       // Matrice A
    int *lda,        // Leading dimension de A
    double *B,       // Matrice B
    int *ldb,        // Leading dimension de B
    double *w,       // Valeurs propres
    double *work,    // Tableau de travail
    int *lwork,      // Taille de work
    int *info        // Information de sortie
);
```

4.2.3 Complexité algorithmique

— Réduction à la forme standard : $O(n^3)$
— Algorithme QR : $O(n^3)$
— Stockage : $O(n^2)$

4.3 Choix du solveur pour notre problème

Bien que la matrice A soit creuse (seulement 5 diagonales non nulles), nous avons opté pour le solveur dense `dsygv` pour les raisons suivantes :

1. **Taille modérée du problème** : Pour $N = 50$, nous avons 2500 degrés de liberté, ce qui donne des matrices de taille 2500×2500. En format dense, cela nécessite environ 50 Mo de mémoire, ce qui est raisonnable pour les ordinateurs modernes.
2. **Simplicité d'implémentation** : Le solveur `dsygv` est plus simple à utiliser que les solveurs creux comme FEAST, qui nécessitent des paramètres de configuration supplémentaires.

3. **Fiabilité numérique** : `dsygv` est un solveur éprouvé de LAPACK, garantissant une grande stabilité numérique.
4. **Extraction de tous les modes** : Contrairement à FEAST qui extrait sélectivement des valeurs propres dans un intervalle, `dsygv` calcule toutes les valeurs propres, ce qui permet une analyse complète du spectre.

Pour des problèmes plus grands ($N > 100$), il serait nécessaire d'utiliser un solveur adapté aux matrices creuses comme FEAST pour des raisons de mémoire et de temps de calcul.

5 Application au problème de la membrane : Résultats et analyse

5.1 Configuration du problème

Nous avons résolu le problème de la membrane non uniforme avec les paramètres suivants :
— Taille du maillage : $N = 50$ points par dimension
— Degrés de liberté totaux : 2500
— Valeurs propres calculées : 10 (les premières)
— Solveur utilisé : `dsygv` (dense)
— Nombre de threads MKL : 4

5.2 Résultats numériques

5.2.1 Valeurs propres et fréquences

Les 10 premières valeurs propres λ_i et fréquences $f_i = \frac{\sqrt{\lambda_i}}{2\pi}$ obtenues sont :

Mode	**Valeur propre λ_i**	**Fréquence f_i (Hz)**
1	2.282654	0.240
2	9.519506	0.491
3	9.850442	0.500
4	18.373903	0.682
5	36.200651	0.958
6	40.813765	1.017
7	46.471895	1.085
8	50.030441	1.126
9	72.231516	1.353
10	82.583877	1.446

TABLE 2 – Les 10 premières valeurs propres et fréquences de la membrane

5.2.2 Temps de calcul

Le solveur `dsygv` a résolu le problème en **8.877 secondes** avec 4 threads. La conversion des matrices creuses au format dense a pris un temps négligeable.

5.2.3 Convergence

Le solveur a convergé en une seule itération (comme attendu pour la méthode QR) avec des résidus nuls à la précision machine (`info = 0`).

5.3 Analyse physique des résultats

5.3.1 Mode fondamental

Le mode fondamental à 0.240 Hz correspond à la vibration la plus basse possible de la membrane. Cette fréquence est influencée par :

— La tension moyenne $p(x, y)$
— La masse moyenne $w(x, y)$
— La présence de l'obstacle central $q(x, y)$ qui augmente légèrement la fréquence

5.3.2 Quasi-dégénérescence des modes 2 et 3

Les modes 2 et 3 ont des fréquences très proches (0.491 Hz et 0.500 Hz). Cette quasi-dégénérescence s'explique par :

1. La géométrie carrée du domaine qui, dans le cas uniforme, donne des modes dégénérés
2. La légère brisure de symétrie due aux coefficients non uniformes $p(x, y)$ et $w(x, y)$
3. L'obstacle central qui affecte différemment les deux modes

5.3.3 Distribution spectrale

La croissance des valeurs propres suit approximativement la loi $\lambda_n \propto n$ (pour une membrane 2D), mais avec des modifications dues aux non-uniformités. L'écart entre valeurs propres successives augmente avec le numéro de mode, ce qui est typique des systèmes vibratoires.

5.4 Analyse de convergence numérique

Nous avons étudié la convergence des valeurs propres en fonction de la finesse du maillage :

N	**DOF**	λ_1	f_1 (Hz)	**Temps (s)**
20	400	2.438390	0.249	0.075
30	900	2.351600	0.244	0.591
40	1600	2.308456	0.242	2.462
50	2500	2.282654	0.240	8.877

TABLE 3 – Convergence de la première valeur propre avec la finesse du maillage

5.4.1 Ordre de convergence

L'erreur relative sur λ_1 entre $N = 40$ et $N = 50$ est d'environ 1.1%. La méthode des différences finies d'ordre 2 donne une convergence en $O(h^2)$, ce qui est confirmé par nos résultats.

5.4.2 Temps de calcul vs précision

Le temps de calcul augmente rapidement avec N (complexité $O(N^6)$ pour la résolution dense), tandis que la précision gagne seulement en $O(N^{-2})$. Il existe donc un compromis optimal entre temps de calcul et précision.

5.5 Visualisation des modes propres

Les cinq premiers modes propres ont été visualisés (voir figures en annexe). On observe :

1. **Mode 1** : Une bosse unique centrée, symétrique
2. **Mode 2** : Un nœud horizontal au centre
3. **Mode 3** : Un nœud vertical au centre
4. **Mode 4** : Deux nœuds perpendiculaires
5. **Mode 5** : Une structure plus complexe avec un anneau nodal

L'obstacle central $q(x, y)$ modifie localement l'amplitude des modes, particulièrement pour les modes pairs qui ont un nœud au centre.

6 Performances et optimisation avec Intel MKL

6.1 Parallélisme et accélération

Intel MKL a permis d'exploiter efficacement le parallélisme multicœur :

— **Accélération** : Le solveur a utilisé 4 threads, réduisant significativement le temps de calcul
— **Échelle** : Pour $N = 50$, le temps de résolution (8.877 s) est raisonnable pour un problème de cette taille
— **Efficacité mémoire** : Malgré l'utilisation du format dense, la mémoire requise (50 Mo) est modeste

6.2 Comparaison avec une implémentation naïve

Une implémentation naïve de l'algorithme QR aurait une complexité similaire mais serait 10 à 100 fois plus lente due à :

1. L'absence de vectorisation AVX-512
2. L'absence de parallélisation multithread
3. L'utilisation d'algorithmes non optimisés pour le cache
4. L'inefficacité dans les opérations BLAS de niveau 3

6.3 Limitations de l'approche dense

Pour notre problème spécifique :

— **Matrice creuse** : La matrice A a seulement 12300 éléments non nuls sur 6.25 millions (0.2% de remplissage)
— **Gaspillage mémoire** : 99.8% des éléments stockés sont des zéros
— **Opérations inutiles** : L'algorithme QR effectue des opérations sur des zéros

Ces limitations deviennent critiques pour $N > 100$, où le stockage dense n'est plus possible.

6.4 Perspective : Solveurs creux pour problèmes plus grands

Pour des problèmes plus grands, nous recommandons :

1. **Format CSR** : Stocker seulement les éléments non nuls
2. **Solveur FEAST** : Extraire sélectivement les valeurs propres désirées
3. **Solveur itératif** : Méthode de Lanczos ou Arnoldi pour les très grands problèmes

7 Conclusions et enseignements

7.1 Synthèse des résultats

Ce projet a démontré avec succès :

1. La modélisation physique d'une membrane non uniforme vibrante
2. La discrétisation par différences finies d'un problème aux valeurs propres
3. L'utilisation efficace d'Intel MKL pour la résolution numérique
4. L'analyse physique des modes de vibration obtenus
5. L'étude de convergence de la méthode numérique

7.2 Enseignements clés

7.2.1 Sur Intel MKL

— **Facilité d'utilisation** : Intégration simple dans un programme C/C++
— **Performances** : Accélération significative grâce au parallélisme
— **Robustesse** : Stabilité numérique éprouvée pour les problèmes de valeurs propres
— **Polyvalence** : Large gamme de solveurs adaptés à différents problèmes

7.2.2 Sur le calcul scientifique

— **Compromis précision/coût** : Un maillage plus fin améliore la précision mais augmente considérablement le coût computationnel
— **Importance du conditionnement** : Les matrices issues de problèmes physiques sont généralement bien conditionnées
— **Validation physique** : Les résultats numériques doivent toujours être interprétés physiquement

7.2.3 Sur la méthode numérique

— **Efficacité des différences finies** : Simple à implémenter et suffisamment précise pour de nombreux problèmes
— **Structure matricielle** : Exploiter la structure creuse est crucial pour les grands problèmes
— **Choix du solveur** : Adapter le solveur à la taille et à la structure du problème

7.3 Perspectives et améliorations possibles

1. **Extension 3D** : Appliquer la même méthodologie à des volumes vibrants
2. **Maillages non structurés** : Utiliser la méthode des éléments finis pour des géométries complexes
3. **Problèmes non linéaires** : Étudier les grandes déformations de la membrane
4. **Couplages multiphysiques** : Interaction avec un fluide environnant
5. **Optimisation de forme** : Trouver la forme de membrane optimale pour une fréquence donnée

8 Dépôt GitHub du projet

Le code source complet, les données et les scripts de ce projet sont disponibles à l'adresse :

`Dépôt github`

Le dépôt contient :
— Le code source C avec Intel MKL
— Les scripts Python de visualisation
— Les données numériques générées
— Les figures produites
— Ce document LaTeX

9 Conclusion

Intel MKL, à travers ses implémentations optimisées de BLAS et LAPACK, constitue un outil puissant pour le calcul scientifique. Ce projet a montré comment résoudre efficacement un problème physique complexe en combinant modélisation mathématique, discrétisation numérique, et résolution haute performance. Les compétences développées sont transférables à de nombreux domaines de l'ingénierie et de la recherche scientifique.

Références

[1] Intel Corporation, *Intel oneAPI Math Kernel Library Developer Reference*, 2024.

[2] Saad, Y., *Numerical Methods for Large Eigenvalue Problems*, SIAM, 2011.

[3] Trefethen, L. N., *Spectral Methods in MATLAB*, SIAM, 2000.

[4] Polizzi, E., *Density-Matrix-Based Algorithms for Solving Eigenvalue Problems*, Phys. Rev. B, 2009.

[5] Bathe, K. J., *Finite Element Procedures*, Prentice Hall, 2014.

[6] Golub, G. H., & Van Loan, C. F., *Matrix Computations (4th ed.)*, Johns Hopkins University Press, 2013.

[7] Trefethen, L. N., & Bau, D., *Numerical Linear Algebra*, SIAM, 1997.

[8] Anderson, E., et al., *LAPACK Users' Guide (3rd ed.)*, SIAM, 1999.

[9] Dongarra, J. J., et al., *A set of level 3 basic linear algebra subprograms*, ACM Transactions on Mathematical Software, 1990.

[10] Intel Math Kernel Library Developer Reference, `https://software.intel.com/content/www/us/en/develop/documentation/mkl-developer-reference-c/top.html`

[11] Netlib BLAS Documentation, `http://www.netlib.org/blas/`

[12] Netlib LAPACK Documentation, `http://www.netlib.org/lapack/`

[13] LAPACK Working Notes, `http://www.netlib.org/lapack/lawns/`