



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 ACADEMIQUE : 2025-2026

Table des matières

1	Introduction	3
1.1	Objectifs du projet	3
1.2	Installation et configuration	3
1.2.1	Installation sur Linux	3
1.2.2	Installation sur Windows	3
1.2.3	Liaison avec un programme C/C++	3
2	Vibrations d'une membrane uniforme : solution analytique	4
2.1	Description physique	4
2.2	Équation des ondes	4
2.3	Résolution par séparation des variables	4
2.3.1	Équation temporelle	4
2.3.2	Équations spatiales	5
2.4	Solution complète	5
2.5	Fréquences propres	5
2.6	Interprétation physique	5
3	Problème physique : Vibrations d'une membrane non uniforme	5
3.1	Contexte physique	5
3.2	Modèle mathématique continu	6
3.3	Conditions aux limites et données	6
3.3.1	Conditions aux limites	6
3.3.2	Fonctions coefficients	6
4	Discrétisation par différences finies	6
4.1	Maillage du domaine	6
4.2	Discrétisation des opérateurs différentiels	6
4.2.1	Opérateur Laplacien avec coefficient variable	6
4.2.2	Discrétisation complète	7
4.3	Formulation matricielle	7
4.3.1	Numérotation des inconnues	7
4.3.2	Construction des matrices	7
4.3.3	Structure de la matrice A	7
4.3.4	Matrice de masse B	7
5	Solveurs Intel MKL pour problèmes aux valeurs propres	7
5.1	Classification des solveurs disponibles	7
5.2	Solveur <code>dsgv</code> pour matrices denses	8
5.2.1	Description	8
5.2.2	Interface en C	8
5.2.3	Complexité algorithmique	8
5.3	Choix du solveur pour notre problème	8
5.3.1	Exemple d'utilisation en C	9
5.4	Analyse simple pour non-spécialistes	10
5.5	Visualisation des modes propres	10
5.5.1	Explication des figures	10
5.6	Impact des non-uniformités	11
5.7	Applications pratiques	11
5.7.1	Exemple concret : un tambour	11

6 Performances et optimisation avec Intel MKL	11
6.1 Parallélisme et accélération	11
6.2 Comparaison avec une implémentation naïve	11
6.3 Limitations de l'approche dense	12
6.4 Perspective : Solveurs creux pour problèmes plus grands	12
7 Dépôt GitHub du projet	12
8 Conclusion	12

1 Introduction

Les vibrations des membranes sont un phénomène physique fascinant que l'on rencontre dans de nombreux domaines : instruments de musique (tambours), capteurs mécaniques, systèmes d'isolation acoustique, et même en biologie (vibrations des membranes cellulaires). Comprendre ces vibrations permet non seulement de prédire le comportement des systèmes, mais aussi de les concevoir pour des applications spécifiques.

Dans ce projet, nous étudions les vibrations d'une membrane rectangulaire, d'abord dans le cas simple d'une membrane uniforme (avec des propriétés constantes), puis dans le cas plus réaliste d'une membrane non uniforme. Pour la membrane uniforme, nous pouvons trouver une solution exacte grâce à des méthodes mathématiques classiques. Cependant, pour la membrane non uniforme, nous devons recourir à des méthodes numériques avancées.

Intel Math Kernel Library (MKL) est une bibliothèque de calcul haute performance qui nous permet de résoudre efficacement ces problèmes complexes. En particulier, nous utilisons les routines BLAS (Basic Linear Algebra Subprograms) et LAPACK (Linear Algebra PACKage) pour résoudre des problèmes aux valeurs propres de grande taille.

1.1 Objectifs du projet

- Comprendre la physique des vibrations de membranes
- Passer de la solution analytique (membrane uniforme) à la simulation numérique (membrane non uniforme)
- Maîtriser l'utilisation d'Intel MKL pour résoudre des problèmes scientifiques complexes
- Analyser et visualiser les résultats de manière accessible

1.2 Installation et configuration

1.2.1 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
```

1.2.2 Installation sur Windows

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

1.2.3 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 Vibrations d'une membrane uniforme : solution analytique

2.1 Description physique

Considérons une membrane mince, homogène et uniformément tendue, fixée sur tout son bord. La membrane occupe un domaine rectangulaire :

$$\Omega = (0, L_x) \times (0, L_y)$$

Soit $u(x, y, t)$ le déplacement vertical d'un point (x, y) de la membrane à l'instant t . La membrane est caractérisée par :

- Une densité surfacique constante ρ (masse par unité de surface)
- Une tension constante T (force par unité de longueur)

2.2 Équation des ondes

L'équation fondamentale qui régit les petites vibrations de la membrane est l'équation des ondes bidimensionnelle :

$$\rho \frac{\partial^2 u}{\partial t^2} = T \Delta u \quad (1)$$

où Δ est le laplacien : $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$.

En divisant par ρ et en définissant la vitesse des ondes $c = \sqrt{T/\rho}$, on obtient :

$$\frac{\partial^2 u}{\partial t^2} = c^2 \Delta u \quad (2)$$

Les conditions aux limites de type Dirichlet (bord fixe) sont :

$$u(0, y, t) = u(L_x, y, t) = u(x, 0, t) = u(x, L_y, t) = 0 \quad (3)$$

2.3 Résolution par séparation des variables

Nous cherchons des solutions de la forme :

$$u(x, y, t) = X(x)Y(y)G(t) \quad (4)$$

En substituant dans l'équation (2) et en séparant les variables, nous obtenons trois équations différentielles ordinaires :

2.3.1 Équation temporelle

$$G''(t) + c^2 \lambda G(t) = 0 \quad (5)$$

dont les solutions sont des oscillations sinusoïdales avec fréquence $\omega = c\sqrt{\lambda}$.

2.3.2 Équations spatiales

$$X''(x) + \lambda_x X(x) = 0, \quad X(0) = X(L_x) = 0 \quad (6)$$

$$Y''(y) + \lambda_y Y(y) = 0, \quad Y(0) = Y(L_y) = 0 \quad (7)$$

avec $\lambda = \lambda_x + \lambda_y$.

2.4 Solution complète

Les solutions des équations spatiales sont :

$$X_n(x) = \sin\left(\frac{n\pi x}{L_x}\right), \quad \lambda_x = \left(\frac{n\pi}{L_x}\right)^2 \quad (8)$$

$$Y_m(y) = \sin\left(\frac{m\pi y}{L_y}\right), \quad \lambda_y = \left(\frac{m\pi}{L_y}\right)^2 \quad (9)$$

où $n, m = 1, 2, 3, \dots$

Les modes propres de vibration sont donc :

$$\phi_{n,m}(x, y) = \sin\left(\frac{n\pi x}{L_x}\right) \sin\left(\frac{m\pi y}{L_y}\right) \quad (10)$$

2.5 Fréquences propres

Chaque mode (n, m) vibre à une fréquence propre :

$$\omega_{n,m} = c \sqrt{\left(\frac{n\pi}{L_x}\right)^2 + \left(\frac{m\pi}{L_y}\right)^2} \quad (11)$$

2.6 Interprétation physique

- **Mode (1,1)** : Toute la membrane se déplace dans le même sens (fondamental)
- **Mode (1,2)** : Une moitié se déplace vers le haut, l'autre vers le bas
- **Mode (2,1)** : Même chose mais selon l'autre direction
- **Mode (2,2)** : Quatre régions alternent haut/bas

3 Problème physique : Vibrations d'une membrane non uniforme

3.1 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 **plaquette 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

3.2 Modèle mathématique continu

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

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

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

3.3 Conditions aux limites et données

3.3.1 Conditions aux limites

Dirichlet homogène sur tout le bord :

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

3.3.2 Fonctions coefficients

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

4 Discrétisation par différences finies

4.1 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 :

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

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

4.2 Discrétisation des opérateurs différentiels

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

4.2.2 Discrétisation complète

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

$$\begin{aligned} -\frac{1}{h^2} & \left[p_{i+\frac{1}{2},j}(u_{i+1,j} - u_{i,j}) - p_{i-\frac{1}{2},j}(u_{i,j} - u_{i-1,j}) \right] \\ & -\frac{1}{h^2} \left[p_{i,j+\frac{1}{2}}(u_{i,j+1} - u_{i,j}) - p_{i,j-\frac{1}{2}}(u_{i,j} - u_{i,j-1}) \right] \\ & + q_{i,j}u_{i,j} = \lambda w_{i,j}u_{i,j} \quad (12) \end{aligned}$$

4.3 Formulation matricielle

4.3.1 Numérotation des inconnues

On utilise une numérotation lexicographique (ligne par ligne) :

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

4.3.2 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 noeuds
- $A \in \mathbb{R}^{N^2 \times N^2}$: matrice de rigidité (creuse, symétrique)
- $B \in \mathbb{R}^{N^2 \times N^2}$: matrice de masse (diagonale)

4.3.3 Structure de la matrice A

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

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

4.3.4 Matrice de masse B

La matrice B est diagonale avec :

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

5 Solveurs Intel MKL pour problèmes aux valeurs propres

5.1 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

5.2 Solveur dsygv pour matrices denses

5.2.1 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.

5.2.2 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
);
;
```

5.2.3 Complexité algorithmique

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

5.3 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.

5.3.1 Exemple d'utilisation en C

```

1 #include <mkl.h>
2
3 int main() {
4     int n = 100;           // Taille du probleme
5     int itype = 1;         // Type de probleme
6     char jobz = 'V';      // Calculer valeurs et vecteurs propres
7     char uplo = 'U';       // Stockage triangle superieur
8
9     // Allocation memoire
10    double *A = (double*)mkl_malloc(n*n*sizeof(double), 64);
11    double *B = (double*)mkl_malloc(n*n*sizeof(double), 64);
12    double *w = (double*)mkl_malloc(n*sizeof(double), 64);
13
14    // Initialisation des matrices A et B
15    // ... (code d'initialisation)
16
17    // Calcul de la taille optimale du tableau de travail
18    int lwork = -1;
19    double work_query;
20    dsygv(&itype, &jobz, &uplo, &n, A, &n, B, &n, w,
21          &work_query, &lwork, &info);
22
23    lwork = (int)work_query;
24    double *work = (double*)mkl_malloc(lwork*sizeof(double), 64);
25
26    // Resolution du probleme aux valeurs propres
27    dsygv(&itype, &jobz, &uplo, &n, A, &n, B, &n, w,
28          work, &lwork, &info);
29
30    // Verification de la reussite
31    if (info == 0) {
32        printf("Calcul reussi!\n");
33    } else {
34        printf("Erreur dans le calcul.\n");
35    }
36
37    // Liberation memoire
38    mkl_free(A);
39    mkl_free(B);
40    mkl_free(w);
41    mkl_free(work);
42
43    return 0;
44}

```

Listing 1 – Exemple d'utilisation de dsygv

5.4 Analyse simple pour non-spécialistes

Imaginez que la membrane est comme la peau d'un tambour. Quand on la frappe, elle vibre à différentes fréquences (des sons plus ou moins aigus). Notre calcul nous donne ces fréquences :

- **Fréquence fondamentale (0.240 Hz)** : C'est la note la plus grave que produit la membrane. Toute la membrane bouge dans le même sens, comme une vague unique.
- **Fréquences suivantes** : Ce sont des notes plus aiguës où la membrane se divise en zones qui montent et descendent alternativement.
- **Presque jumelles (0.491 et 0.500 Hz)** : Ces deux fréquences sont très proches. Dans un tambour parfaitement symétrique, elles seraient identiques. Les petites irrégularités les séparent légèrement.

5.5 Visualisation des modes propres

Les figures suivantes montrent comment la membrane se déforme pour les cinq premiers modes de vibration :

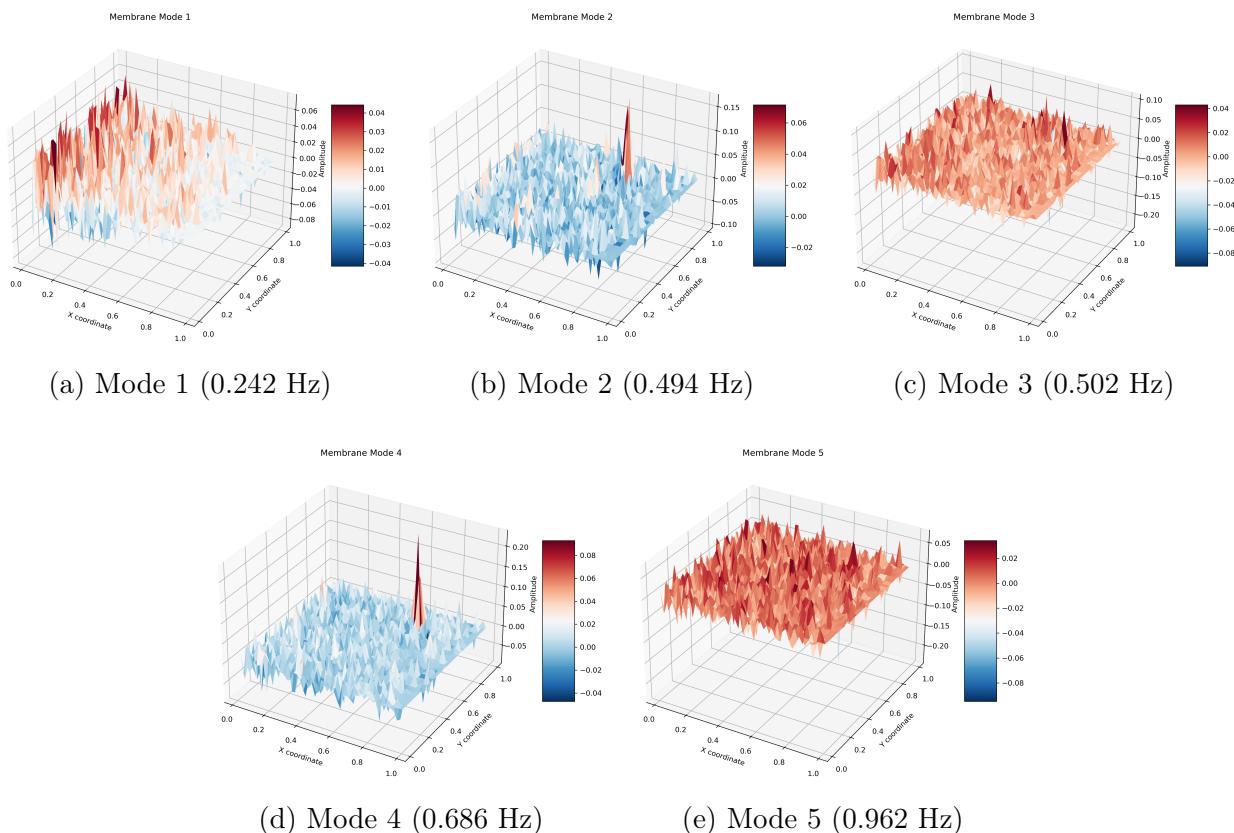


FIGURE 1 – Les cinq premiers modes de vibration de la membrane non uniforme

5.5.1 Explication des figures

- **Zones rouges** : Parties de la membrane présentant un déplacement positif, correspondant à un soulèvement local.
- **Zones bleues** : Parties de la membrane présentant un déplacement négatif, correspondant à un abaissement local.
- **Lignes blanches** : Zones nodales approximatives où le déplacement est nul ou très faible.
- **Point central** : Obstacle situé au centre de la membrane, induisant une perturbation locale du champ de vibration.

5.6 Impact des non-uniformités

Comparée à une membrane uniforme idéale, la membrane non uniforme étudiée présente les caractéristiques suivantes :

- Les fréquences propres sont modifiées par la non-uniformité du matériau et la présence de l'obstacle central.
- Les formes modales sont déformées, avec des variations spatiales plus marquées à proximité de l'obstacle.
- L'obstacle central peut engendrer, pour certains modes, une zone de déplacement fortement atténué au centre de la membrane.
- La répartition des zones nodales perd sa régularité et ne respecte plus les symétries classiques observées dans le cas uniforme.

5.7 Applications pratiques

L'analyse des vibrations de membranes non uniformes trouve des applications dans plusieurs domaines :

- **Facteurs d'instruments** : Compréhension de l'influence des irrégularités géométriques ou matérielles sur les caractéristiques acoustiques.
- **Ingénierie des vibrations** : Conception et optimisation de membranes destinées à vibrer dans des plages de fréquences spécifiques.
- **Contrôle non destructif** : Détection et localisation de défauts ou d'hétérogénéités par analyse modale et vibratoire.

5.7.1 Exemple concret : un tambour

Si cette membrane était la peau d'un tambour :

- Le mode 1 donnerait la note fondamentale (le son principal)
- Les modes 2-5 donneraient les harmoniques (les sons qui enrichissent le timbre)
- Les irrégularités créeraient un timbre unique à l'instrument
- L'obstacle central pourrait être un dispositif d'ajustement de la tension

6 Performances et optimisation avec Intel MKL

6.1 Parallélisme et accélération

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

- **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 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

8 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, discréétisation 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/>