LINMA1170 – Projet : Optimisation de forme d'un diapason

Victor Lepère, Alexandre Nicolas

19 mai 2023

Résumé

Ce rapport rassemble nos différentes analyses et conclusions vis-à-vis du projet du cours d'Analyse Numérique dispensé par Pr. Jean-François Remacle. Il nous a été demandé de concevoir une pièce mécanique dont la fréquence propre correspond à une note donnée. La note de musique que nous avons choisie est le La de la sixième octave (A6) qui correspond à une fréquence de 1760 Hz.

Optimisation de la forme

Afin que notre diapason résonne selon la note voulue, il nous a fallu résoudre l'équation $f_0(\beta_1, ..., \beta_m) - 1760 = 0$ où β_i sont les paramètres de design de la forme et $f_0(\underline{\beta})$ représente la fréquence fondamentale calculée sur la géométrie selon ces paramètres.
Pour ce faire, plusieurs algorithmes de recherche de racine existent dont notamment la méthode de la bissection [1]. La difficulté dans l'utilisation de ces méthodes réside dans le choix de l'intervalle ou des itérés initiaux et ce pour plusieurs raisons. Premièrement, la haute magnitude de notre fréquence réduit le nombre de combinaisons de paramètres de design possible. Les paramètres optimaux conduisent souvent à des formes de diapason atypiques (des diapasons très courts et épais). Par ailleurs, le raffinement du maillage induit une diminution de la fréquence propre calculée. Ainsi, si nous disposons par exemple d'un intervalle initial qui contient la solution pour la méthode de dichotomie, il n'est pas garantit que cet intervalle soit toujours valide pour une taille de maillage moindre.
Ces difficultés s'intensifient pour le calcul des fréquences harmoniques dont la plus petite s'élève à 3520 Hz.

Méthode de la bissection multivariée

Afin de pouvoir jouer sur m paramètres de design simultanément, nous avons implémenté une version légèrement modifiée de la méthode de dichotomie classique (voir la fonction `polyDichotomous` dans `dichotomous.h`). Cette dernière peut être étendue pour agir sur plusieurs variables en utilisant une approche itérative de la manière suivante. A chaque paramètre est associé un intervalle dans lequel on suppose que la valeur optimale réside. Ainsi, à chaque itération, le paramètre est mis à jour de sorte à ce que la fréquence propre calculée sur la nouvelle géométrie se rapproche de la fréquence cible.

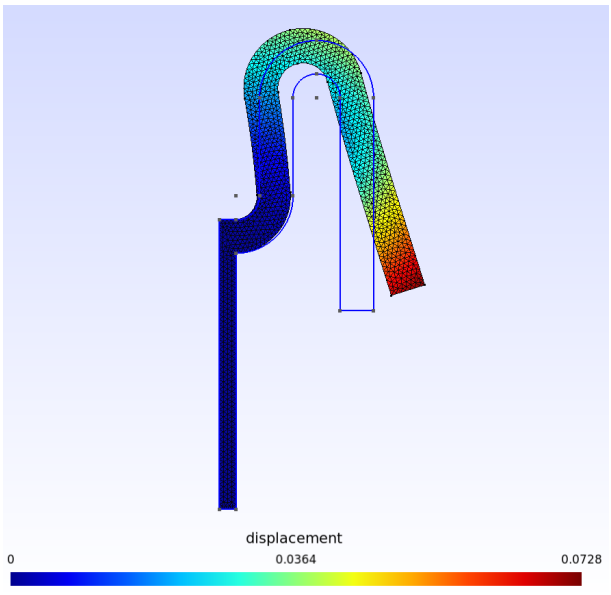


FIGURE 1 – Un diapason cocasse.

Grâce à cet algorithme simple, il est possible de créer des diapasons aux géométries atypiques comme l'illustre la Fig. 1. La théorie des éléments finis et la résolution du problème aux valeurs propres généralisé nous permettent de calculer pour un maillage constitué de 2551

1. `https://fr.wikipedia.org/wiki/M\%C3\%A9thode_de_dichotomie`

éléments que la fréquence propre de ce diapason vaut 1759.861 Hz, ce qui est proche de notre note. Le calcul est effectué sur la moitié du diapason auquel nous avons imposé une symétrie pour des raisons de performances explicitées dans la section suivante.

Toutefois, il faut noter que la méthode de la bissection étant assez simpliste, son ordre de convergence est seulement linéaire. L'erreur n'est donc réduite que d'un facteur constant à chaque itération. De plus, l'interdépendance des paramètres dans notre version multivariée peut ralentir la méthode. En contrepartie, l'espace des solutions explorées est plus large que celui de la méthode classique. Ainsi, il a nécessité 28 itérations pour produire le résultat de la Fig. 1. On comprends la nécessité d'avoir un code efficace pour calculer la fréquence propre sur une géométrie donnée.

Nous voulions initialement calculer la plus petite fréquence harmonique (A7) en appliquant la méthode de dichotomie sur la longueur de la tige extérieure de ce diapason après l'avoir appliqué sur la longueur intérieur. Seulement, il s'avère que les deux pôles de vibrations sont dépendants et une telle approche n'est pas réalisable (la longueur de la tige extérieure influence la fréquence fondamentale). Une autre approche est d'incorporer le calcul de la fréquence harmonique au sein même de celui de la fréquence fondamentale. La fonction objectif à annuler devient alors $g(\underline{\beta}) = |f_0(\underline{\beta}) - 1760| + |f_1(\underline{\beta}) - 3520|$. L'apparition des valeurs absolues rend inutilisable la méthode de la bissection qui se base sur le principe des valeurs opposées : si $f(a)$ et $f(b)$ sont de signes opposés, alors il existe au moins une racine de la fonction entre a et b. Nous nous sommes alors tourner vers les algorithmes de minimisation de fonction ne nécessitant pas le calcul des dérivées de la fonction objectif (inconnues ici). L'une d'entre elles, connue sous le nom de méthode du simplexe ou méthode de Nelder-Mead, est présentée dans le paragraphe ci-après.

Méthode de Nelder-Mead

Cette méthode d'optimisation non linéaire permet de minimiser une fonction dans un espace multidimensionnel au moyen d'une figure géométrique formée de points de l'espace de recherche nommée simplexe. La fonction

```
1 double *nelderMead(double(*f)(double *), const double *initialParams, const
      double *lBounds, const double *uBounds, int nmax, double tol);
```

définie dans `nelderMead.h` est une implémentation en language C de cet alogorithme réalisée pour ce projet. Il s'agit d'une version légèrement modifiée de l'algorithme usuel [2] qui inclut la possibilité de borner les différents paramètres pour ne pas créer des géométries indéfinies.

Il suffit alors d'appliquer la méthode sur la fonction objectif $g(\underline{\beta})$ pour obtenir les paramètres de design recherchés. Cependant, après de nombreuses tentatives infructueuses, nous nous sommes vite rendus compte que notre forme de diapason ne permettait physiquement pas d'obtenir comme première et seconde fréquence propres 1760 Hz et 3520 Hz à cause du trop gros écart entre ces deux valeurs. En effet, nous avons remarqué que pour des jeux de paramètres générant une fréquence fondamentale propre de 1760 Hz, le rapport entre le seconde fréquence propre et la première variant entre 1.3 et 1.7.

C'est pourquoi nous avons décider d'employer la méthode de Nelder-Mead pour obtenir comme seconde fréquence propre la fréquence correspondant à la quinte [3] pure de notre La. Nous avions initialement essayer d'obtenir la tierce [4] majeure, soit $f_1 \to \frac{5}{4} f_0 = 2112$ Hz. Cependant, la méthode n'a pas convergé pour les mêmes raisons que l'octave supérieure. La quinte de A6 correspond à la note E#7 (Mi dièse) ou $\frac{3}{2}1760 = 2640$ Hz en termes de fréquence. En

2. https://fr.wikipedia.org/wiki/M|\%C3\%A9thode_de_Nelder-Mead
3. https://fr.wikipedia.org/wiki/Quinte
4. https://fr.wikipedia.org/wiki/Tierce_(musique)

appliquant la méthode sur la fonction $g(\underline{\beta}) = |f_0(\underline{\beta}) - 1760| + |f_1(\underline{\beta}) - 2640|$ pour un maillage de 1911 éléments, nous obtenons la forme représentée sur la Fig. 3.

Convergence des algorithmes

Dans l'optique de se donner une idée de l'efficacité des algorithmes de Nelder-Mead et de la bissection, nous avons étudié dans cette section le nombre d'itération de ces deux méthodes. Les résultats obtenus sont repris à la figure 2, sur laquelle nous pouvons lire le nombre d'itération des méthodes en fonction de la taille du maillage utilisé.

Dans un premier temps, nous pouvons observer que les bâtonnets correspondant à la méthode de la bissection semblent garder une valeur oscillant entre 41 et 49, ce qui signifie que le nombre d'itération est à peu près constant quelles que soient les dimensions du problème. Ce phénomène est logique, puisque cette méthode se concentre sur la recherche d'une racine au sein d'un intervalle, qu'elle divise par deux à chaque itération, plutôt que sur la discrétisation en elle-même du problème.

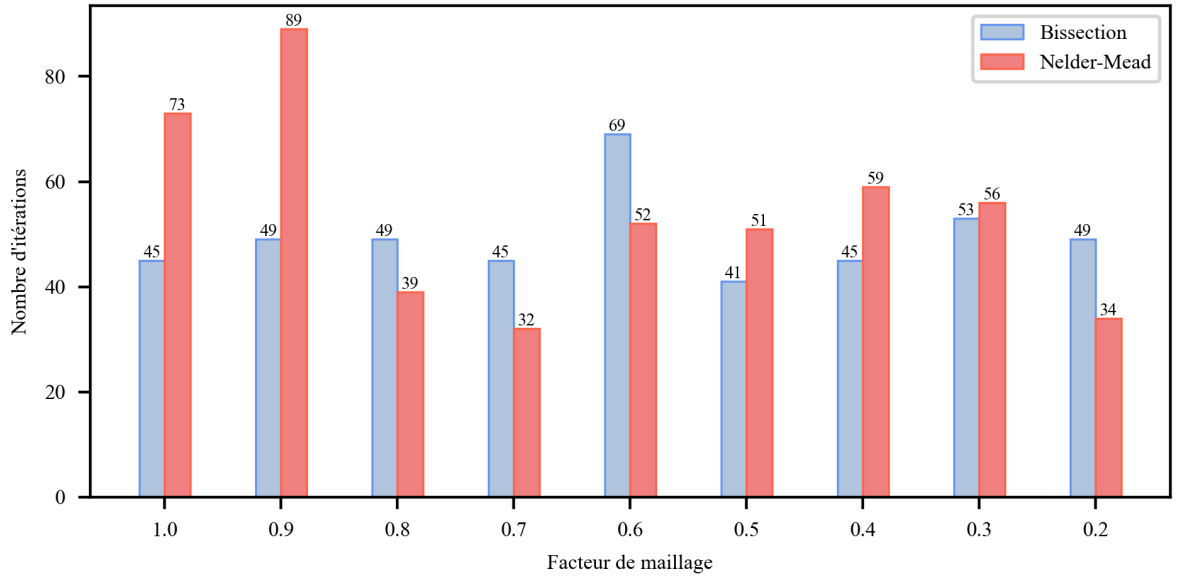


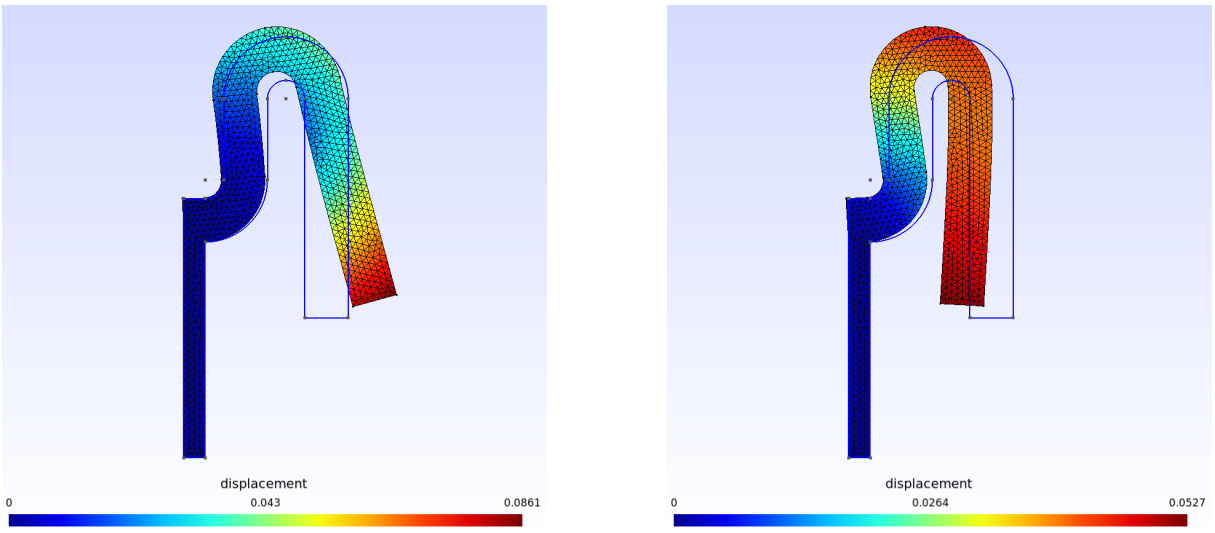
FIGURE 2 – Nombre d'itérations des méthodes.

Ensuite, si nous nous concentrons sur les résultats de la méthode de Nelder-Mead, nous aperçevons cette fois-ci que le nombre d'itérations décroît avec la taille du maillage. Ceci est dû au fonctionnement intrinsèque de la méthode de Nelder-Mead : cette dernière minimise itérativement une fonction objectif. Or, lorsque le maillage contient plus d'éléments, nous pouvons obtenir une meilleure représentation numérique de notre problème et ainsi approcher notre fonction objectif plus efficacement, ce qui induit un nombre d'itérations réduit.

Usinage

Au sein des sections précédentes, nous avons présentés plusieurs formes de diapason satisfaisant des contraintes de design précises (Fig. 1, 3). Le tableau 2 synthétise les valeurs des différents paramètres obtenues.

Nous avons aussi appliqué la méthode de la bissection sur la géométrie fournie dans l'énoncé du projet (diapason classique) et retranscrit les résultats dans la table.



(a) $f_0 = 1759.577\,\text{Hz}$ (b) $f_1 = 2639.249\,\text{Hz}$

FIGURE 3 – Visualisation des modes propres.

	Epaisseur $[mm]$	Manche $[cm]$	Fourche interne $[cm]$	Fourche externe $[cm]$
Custom	4.25	3.25	1.25	≈ 2.7
Custom + Quinte	7	≈ 3.47	≈ 1.3	≈ 3.52

TABLE 1 – Jeux de paramètres pour la forme personnalisée.

	Epaisseur $[mm]$	Manche $[cm]$	Fourche $[cm]$
Classique	≈ 5.27	1.25	≈ 3.875

TABLE 2 – Jeu de paramètre pour la forme classique.

On observe que les différents algorithmes convergent systématiquement vers des diapasons de petites tailles. Nous avons essayé de générer des plus grands formats de diapason en adaptant les bornes de recherche en conséquence. Les algorithmes n'ont malheureusement pas convergé car une petite hauteur semble être une contrainte intrinsèque à tout objet résonnant à la fréquence recherchée, qui est assez élevée. De plus, multiplier proportionnellement l'ensemble des paramètres par un facteur constant n'est pas une solution car la fréquence fondamentale de la nouvelle forme ne correspond plus à 1760 Hz. Il s'agit ici de la piste d'amélioration principale du projet.

Performances

Clairement, notre implémentation du calcul des fréquences propres est axée sur la rapidité d'exécution. Pour ce faire, nous avons exploité deux stratégies particulières. La première est l'implémentation de la symétrie. En effet, le code a été adapté pour pouvoir traité la symétrie sur n'importe quelle géométrie. Pour l'utiliser, il suffit d'ajouter un groupe physique contenant l'ensemble des courbes définissant l'axe de symétrie et le nommer `symmetry`. Par exemple,

```
1 int symmetryCurveTags[1] = {15};
2 gmshModelAddPhysicalGroup(1, symmetryCurveTags, 1, -1, "symmetry", &ierr);
```

Dès lors, le déplacement vertical de l'ensemble des noeuds se trouvant sur l'axe de symétrie est imposé nul. Ceci permet d'exclure d'une part d'exclure les modes de vibrations asymétriques inaudibles du diapason. Nous pouvons ainsi nous débarrasser des opérations de déflation associées au sein de l'algorithme de base (devoir 3). D'autre part, le nombre de noeuds du maillage est divisé par 2. Cette division permet d'accélérer le processus de manière impressionnante. En effet, pour une taille de maillage fixée, le coût de calcul le plus important lors de la résolution du problème aux valeurs propres généralisé est induit par la factorisation LU de $K^{n\times n}$. L'ordre de complexité de ce calcul est $\mathcal{O}(n^3)$. Ainsi, si nous réduisons de moitié le nombre de noeuds, le temps de calcul devient d'ordre $\mathcal{O}(\frac{n^3}{8})$. Dès lors, nous pouvons nous attendre à ce que l'implémentation s'effectue près de 8 fois plus rapidement ! La Fig. 4 montre que cette hypothèse est vérifiée pour un nombre de noeuds suffisamment grand [5]. Il est alors possible pour un même temps d'exécution de calculer des modes de vibration selon un maillage davantage raffiné et donc de diminuer l'erreur de discrétisation.

Le seconde astuce employée pour obtenir un code efficace est l'utilisation de librairies optimisées d'algèbre linéaire telle que l'interface C de LAPACK (Linear Algebra Package) connue sous le nom de LAPACKE. La bibliothèque propose la fonction `LAPACKE_dsygv` qui permet de résoudre le problème aux valeurs propres généralisé $Kx = \lambda Mx$. Comme en témoigne la Fig. 4, le temps d'exécution est diminué à nouveau drastiquement réduit et le code s'exécute entre 6 à 7 fois plus rapidement pour des maillage raffiné.

Ces deux optimisations combinées nous permettent d'obtenir un code très efficace pour calculer les fréquences propres qui nous permet de réduire à quelques secondes la durée de chaque itération de la méthode de la bissection.

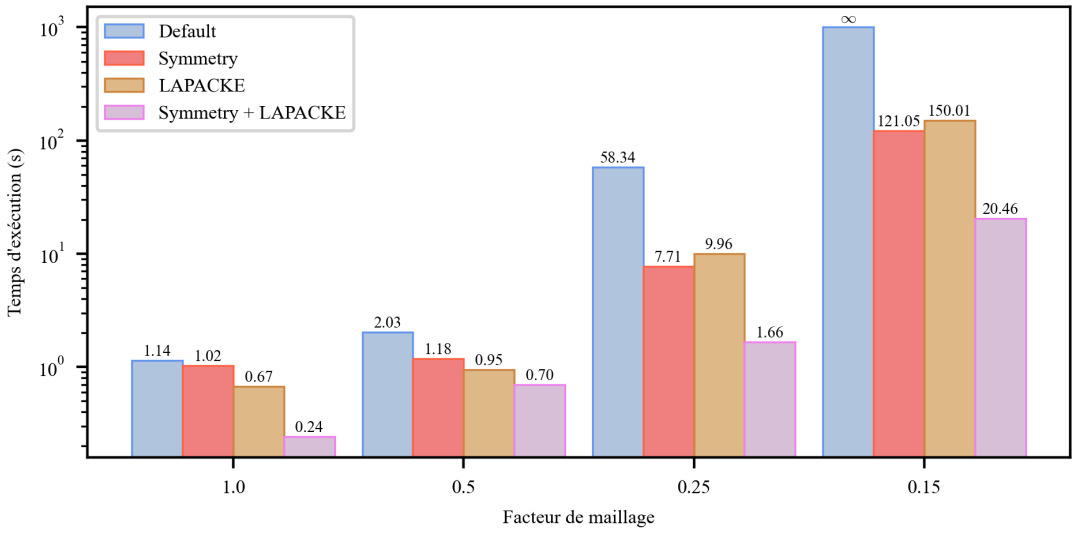


FIGURE 4 – Temps de calcul des fréquences propres

Implémentation

L'ensemble des indications relatives à la compilation et l'exécution du code peuvent être trouvées dans le `README.md` associé au projet. Nous conseillons d'ouvrir ce dernier via une extension de *preview* de *MarkDown* pour apprécier pleinement du soin apportée à la mise en forme.

5. La notation ∞ sur la figure correspond à un temps d'exécution supérieur à 10 minutes.