Rapport : Projet Analyse Numérique

Alexandre Dewilde

15 mai 2023

Les détails de comment lancer les scripts, etc sont dans le readme

1 Objectif de base

L'objectif de base consistait à trouver un diapason qui sonnait à la fréquence 1567.98Hz correspondant à la note **G6**, plusieurs approches ont été considérés détaillé ci-dessous.

1.1 Bisection

1.1.1 Description

La bisection est un algorithme très simple et très efficace qui permet de trouver une valeur sur un intervalle qui est croissante ou décroissante en coupant l'intervalle au milleu.

Lorsque l'on dimensionne un diapason on remarque de manière empirique que si l'on augmente/diminue la longueur des branches du diapason sa fréquence diminue/augmente, ce qui indique que si on fixe les autres paramètres et on a une fonction décroissante, on résume le problème alors à trouver $(f(L) - f_{target}) = 0$.

1.1.2 Convergence

La bisection converge très rapidement, en temps logarithmique car l'on coupe chaque fois l'intervalle au milleu, pour notre diapason celle-ci converge en moins de **10 itérations** pour une différence absolue entre f et f_{target} inférieur à 1.

La convergence théorique corrèle selon wikipédia [1], la complexité est borné par

$\lceil log_2(\frac{\epsilon_0}{\epsilon}) \rceil$.

1.1.3 Résultats

Pour r1, r2 et e fixé à 6e-3, 11e-3, 38e-3 au départ, la bisection converge vers une longueur de 0.038638m, ce qui donne pour le premier mode symétrique une fréquence de 1568.393Hz, à moins d'un hertz de la fréquence cible.

1.1.4 Conclusion

On a un joli diapason bien dimensionné avec un manche de longueur presque égale à ses branches, dans un temps très court grâce à la convergence rapide de la bisection.

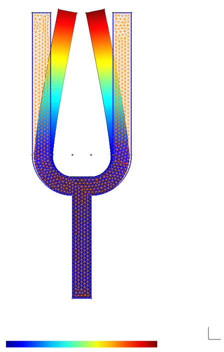


FIGURE 1 – Premier mode symétrique, longueur calculé via bisection, autres paramètres fixés, meshsize=0.2

1.2 Ternary search

1.3 Description

La bisection est un algorithme limité au fonction croissante/décroissante, si l'on veut considérer d'autres contraintes comme le poids constant, la fonction ne sera plus forcément croissante/décroissante. Il sera donc plus intéressant de s'intéresser à des fonctions convexes qui ont la propriétés que la somme de deux fonctions convexes sont convexes.

L'algorithme de recherche ternaire est un algorithme très semblable à la bisection mais qui trouve le minimum d'une fonction convexe, elle évalue la fonction en deux endroit distinct pour choisir un intervalle de recherche.

1.3.1 Fonction de coût

Il faut rendre notre fonction de base convexe, pour cela il y a différente *fonction de coût* comme la MSE $(f - f_{target})^2$ ou la MAE $|f - f_{target}|$, les deux sont similiaires mais la MSE pénalise plus fortement les grosses erreurs du à son facteur carré.

1.3.2 Convergence

La ternary search a elle aussi une convergence logarithmique pour trouver un diapason avec $|f - f_{target}| < 1$ il faut moins de 20 itérations, ce qui est plus que la bisection, du au fait que l'intervalle est coupé en trois et non en deux.

1.3.3 Résultats

Avec un meshsize de 0.2 pour r1=6e-3, r2=11e-3, e=28e-3 fixés on a l=0.038702

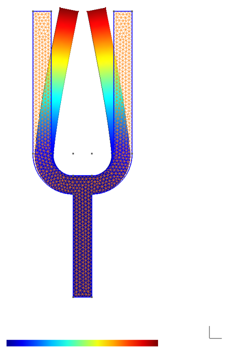


FIGURE 2 – Premier mode symétrique pour ternary search, meshsize=0.2

1.4 Descente de gradient

1.4.1 Description

Les méthodes précédente sont trés limité elles ne prennent qu'un paramètre en compte et les adapter à plusieurs paramètres ne fonctionnerait pas forcément car il faudrait que les fonctions reste (dé)croissante pour bisect et convexe pour ternary search.

La descente de gradient résoud ce problème, il peut minimiser plein de fonctions de type différents, il peut passer au dessus des minimas locaux, la fonction peut être de toutes formes, c'est un algorithme itératif très simple pour minimiser/maximiser une fonction, il est de cette forme $\beta^{k+1} = \beta^k - \alpha \cdot \nabla f(\beta^k)$ où α représente le "pas".

1.4.2 Approximation du gradient

Dans la descente de gradient il faut à chaque itérations calculer un vecteur gradient, pour cela une façon simple est la méthodes des différences finis [2] La différence finis centré est exprimé de cette façon $f'(x) = \frac{(f(x-h)+f(x+h))}{2h}$, la méthode à une précision d'ordre $O(h^2)$, attention que lorsque le h est trop petit, il y a l'erreur de division sur les flottants.

1.4.3 Fonction de coût

Il faut une fonction à optimiser comme pour ternary search 1.3.1, les mêmes peuvent être utilisé. Pour le descente de gradient la MSE sera utilisé.

1.4.4 Boundaries

Il est très important de définir des boundaries corrects pour la descente de gradient, sinon ils pourraient avoirs des paramètres qui vont dans les négatifs, d'autres qui exploserait, pour cette géometrie il est très important que r2 soit plus grand que r1 car la largeur des branches est la différence de r2 et r1. Les paramètres seront limités entre 1e-3 et 1e-1 pour garder des dimensions corrects.

1.4.5 Gérer les boundaries

Une manière simple de gérer les boundaries est d'empecher les paramètres de descendre/monter au dessus par exemple avec $max(min(f, 1e-1), 1e-3)$

1.4.6 Convergence

La descente de gradient converge plus ou moins rapidement en fonction du α choisis, si α est trop petit cela mettra trop de temps à converger et s'il est trop grand ils pourraient diverger.

Ici avec un $\alpha = 0.1$ on converge pour une différence absolue inférieur à 1 en moins de 40 itérations.

mais la descente de gradient souffre de gros défauts

— Instable
— Nécessite d'approximer un gradient
— Paramètre α qu'il faut choisir, si trop petit peut tomber dans un minimum local et trop grand peut diverger

1.4.7 Résultats

Pour un meshsize de 0.2, on obtient pour r1, r2, e, l : 0.00364276491848620695, 0.02209637632029602672, 0.03911024349560879299, 0.07758227182407308187. Ici on remarque une difference de 10 hertz quand on passe du maillage symmetrique au mallaige classique, cela peut être du au proportion de largeur de ce diapason.

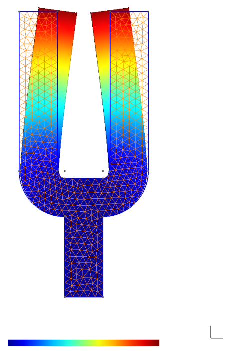


FIGURE 3 – Diapason pour un meshsize de 0.2 obtenu via descente de gradient

1.5 Différentes méthodes via scipy

1.5.1 Description

Pour pouvoir tester une plus grande variétés d'algorithmes sans devoir tous les implémenter, la suite du travail est fait avec la libraire scipy qui permet d'utiliser des méthode d'optimisation plus poussé et meilleur que la descente de gradient et sont SOTA en optimisation.

1.5.2 Fonction de coût et boundaries

Les mêmes fonctions de coût et boundaries sont utilisé que pour la descente de gradient 1.4.4, sauf qu'on fixera r1 par facilité pour ne pas devoir gérer la contraintes $r1 < r2$

1.5.3 Résultats

En utilisant la méthode "Nelder mead" pour un meshsize de 0.2 et en fixant r1 à 0.006, les paramètres suivants donne un diapason sonnant à la fréquence cible.

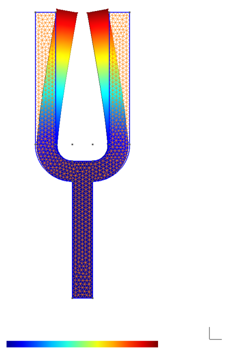


FIGURE 4 – Premier mode symmetrique, meshsize=0.2, dimensions obtenu avec "Nelder mead"

2 Imposer les modes suivants à des harmoniques

Il faut noter la haute fréquence de la note a rendu plus complexes les calculs. À fin de simplifier l'optimisations les calculs sont fait sur un maillage de 0.3.

2.1 Première approche

La première approche *naive* a été de gardé la même géométrie mais de changer la fonction de coût, qui prends en compte les harmoniques suivantes, $\sum |freq_i - i * f_{target}|$, malheuresement ça ne marche pas.

2.2 Diapason multi-couches

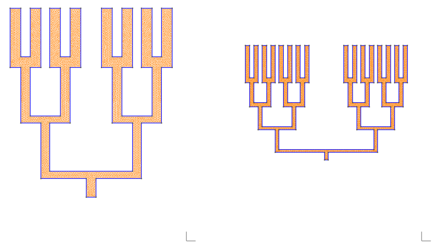
La seconde approche un diapason multi-couches approche proposé dans [3], pour cette géométrie, 2 approches sont testé dans la suite.

2.2.1 Définition de la géométrie

À fin de simplifier au plus la géométrie, les cotés arrondis ont été remplacé par des cotés à angle droit, et plein de nouveau paramètres ont été définis pour permettre plus de liberté à l'optimisation

- Longueur et largeur du manche (lh, wh)
- Largeur des branches par étages (d)
- décalage supplémentaire entre la branche du précédent étages et les branches de l'étage current (dec)
- longueur entre la base des branches et le le points le plus élévé du milleu des deux branches (h)
- longueurs des branches par étages (l)

Les différents paramètres permettent une géométrie très libres.



(a) (b)

FIGURE 5 – Diapasons a 3 et 4 couches

2.2.2 Fonction de coût

La MAE est adaptable ici pour les n modes on aurait alors $\sum |f_i - i * f_{target}|$

2.2.3 Boundaries

Pour rester dans des dimensions raisonables on peut prendre pour les largeurs des dimensions entre (1e-3 et et 1e-2) et (1e-2 1e-1) pour les longueurs.

2.2.4 Bisection pour affiner les résultats

Lorsque l'on prends des maillage plus fin l'optimisation vas diverger plus facilement vers une valeur plus haute alors que souvent il suffit d'adapter les longueurs des branches couches par couches, en effet on constate empiriquement que augmenter les longueurs des branches inférieur à un impact très faible sur les fréquences des couches supérieurs, on peut donc mettre à jour les longueurs des branches du dernier niveau jusqu'au premier niveau, attention que lorque l'on augmente le nombre de couche cela marchera moins bien.

On prendra donc un maillage plus haut par exemple 0.5 sur lequel on fera l'optimisation, on utilisera la bisection avec un maillage plus fin pour affiner les résultats.

Attention que cela peut empirer les choses, il faudra donc comparer les 2 résultats et prendre le meilleur.

2.2.5 Résultats

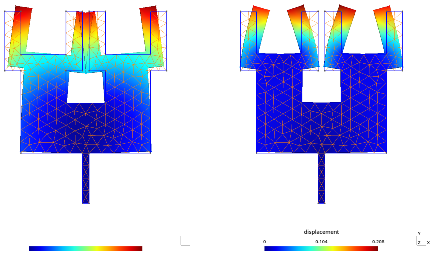
Avec un maillage de 0.3, on a comme paramètre pour un diapason à deux étages

- lh 0.06595566225750074, wh 0.009164195816166532
- d : 0.05929252977121609, 0.021409268854274743
- dec : 0, 0
- 0.06581162483843764, 0.021227934873379272
- 0.042379960230197806, 0.05744559880222302

Il nous donne des fréquences de f1 = 1570Hz pour le mode fondamentale et f2 = 3161Hz ce qui est assez proches des harmoniques

2.2.6 à plus de 3 couches

Cette méthode pourrait être généralisé à n couches pour n modes.



(a) (b)

FIGURE 6 – Mode fondamentale (a) et mode suivant (b) pour le diapason à deux étages (meshsize=0.3)

3 Contraintes sur la géométrie

Pour faciliter l'optimisation les différents calculs sont fait sur un diapason simple mais les principes peuvent être applique sur un diapason multi couche

3.1 Description

On peut vouloir imposer différentes contraintes sur notre géométrie comme un poids constant, une taille minimum

3.2 Poids constant

En 2 dimensions le poids constant équivaut à l'air constante multiplié par l'épaisseur, on cherchera donc a avoir une surface constante

3.2.1 Longueur du manche

Les conditions frontières du manche étant fixé à 0, celui-ci à peu d'impact sur la fréquence, on peut donc trouver les bons paramètres r1, r2, l puis adapter la longueur du manche en fonction de l'air visé.

Il est alors possible de procéder de cette façon :
- Bisection sur la longueur des branches pour avoir la bonne fréquence
- Bisection sur la longueur du manche pour avoir la bonne surface

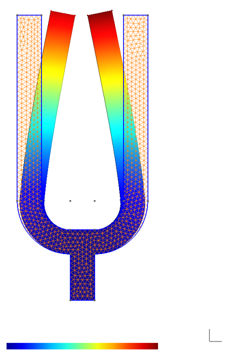


FIGURE 7 – Diapason avec une aire de $5e^{-4}m^2$, en faisant une bissection sur le manche

Mais dans le monde réel, cette méthode naïve ne fonctionnerait peut être pas puisqu'ici on fait l'hypothèse d'un mouvement nul pour les bords du manche.

3.2.2 Pénalité sur la différence d'aire

Une approche pour résoudre ce problème est de considérer une pénalité pour la contrainte de taille, on peut faire ça en ajoutant à la fonction de coût un terme à la fonction tel que la différence absolue, ou la différence carré entre l'aire actuel et l'air voulu.

4 Animation des modes propres

Plusieurs animations se trouve dans le readme.

5 Rapidité du code

5.1 Maillage symmetrique

La première grosse amélioration est le maillage symmétrique, comme précisé dans l'énoncé du projet on peut modéliser le diapason de façon symmetrique et imposer les conditions à 0 sur l'axe symmétrique, cela donne plusieurs avantage de rapidité
- Plus petit mesh donc moins de calcul
- 2x moins de valeurs propres à calculer car l'on a éliminé les modes asymétrique qui ne nous intéressent pas

K modes sym	Temps sym	temps normal
2	0.43s	0.58s
4	0.56s	1.37s
8	1.36s	6.56s

5.2 Renumérotation et matrice bande

Pour pouvoir effectuer l'inversion de la matrice K, on peut renuméroter le noeuds de cette matrice pour avoir une matrice bande avec une toute petite bande, ici l'algorithme de renumérotation utilisé est reverse cucktill mckee qui permet de renuméroter les noeuds. Les chiffres ont déja été donné dans le devoir 3.

5.3 Calcul de la matrice $K^{-1}M$

Pour calculer cette matrice, une façon naïve est d'abord d'inverser la matrice, puis d'effectuer la multiplication matricielle, hors la multiplication est une opération coûteuse en $\mathcal{O}(n^3)$, une façon d'accélérer est de faire la multiplication matricielle lors de l'inversion de la matrice ce qui coûte beaucoup moins cher.

5.4 Routine BLAS

5.4.1 Calcul des valeurs propres

Pour le calcul des valeurs propres plusieurs opérations vectoriel peuvent être remplacé par des routines blas ;
- La normalisation, on peut calculer la norme graçe à cblas_dnrm2 et appliquer la division graçe à cblas_dscal

— On peut calculer le inner/dot product graçe à cblas_ddot
— La multiplication matricielle vecteur peut être fait graçe à la routine cblas_dgemv
— On peut faire la déflation avec la routine cblas_dger

Références

[1] https://en.wikipedia.org/wiki/Bisection_method#Analysis

[2] https://fr.wikipedia.org/wiki/M%C3%A9thode_des_diff%C3%A9rences_finies

[3] Note : Arbitrary periodical mechanical vibrations can be realized in the resonant state based on multiple tuning fork structure