Mathieu MARÉCHAL

Simulation Numérique II

Nom de code : Dém***ez-vous !

Compte-rendu des projets

Encadrants : C. Desjouy, S. Raetz

L2 SPI

2018 - 2019

Introduction

Table des matières

Projet 1 La balle

L'objectif de ce projet est de modéliser la trajectoire d'une ou plusieurs balles en prenant en compte les rebonds. La consigne divise le travail en plusieurs étapes qui consistent à, tout d'abord, calculer et animer la trajectoire sans rebonds d'une balle, puis de prendre en compte les rebonds. Enfin, le programme nécessite d'être capable de calculer plusieurs trajectoires à la fois. J'ai donc décidé de procéder comme cela, en rajoutant des étapes : j'ai tout d'abord cherché à calculer une trajectoire simple et sans animation. Ensuite, j'ai créé les codes successifs dans des fichiers séparés, en rajoutant successivement : les animations, les rebonds, et enfin, plusieurs balles.

1.1 Calculer la trajectoire

Tout d'abord, les premières constantes dont va dépendre la trajectoire sont crées :
— l'angle initial, θ_0 nommé `a_0`
— la vitesse initiale v_0, notée `v_0`
— plus tard, le temps de départ t_0 (`t_0`) sera utilisé
Ensuite, on implémente les équations déterminant la position du projectile en fonction des constantes précédentes :

$$\begin{cases} d_x = & v_0 t cos(\theta) + d_{0_x} \\ d_y = & -\frac{1}{2} g t^2 + v_0 t sin(\theta) + d_{0_y} \end{cases}$$

Il est maintenant possible de calculer la trajectoire mais la première question qui se pose est : quand doit-on arrêter de tracer la trajectoire ? En effet, si on trace la trajectoire sur un temps `t_balle` fixé, on observera la trajectoire avoir une position d_y négative, avec une pente très élevé, ce qui n'a pas du tout la forme d'une parabole. C'est le premier problème que j'ai rencontré. C'est donc quand le projectile touche le sol qu'il faut arrêter de calculer la trajectoire, quand sa position en y vaut $0 \Leftrightarrow d_y = 0$. Pour cela, il faut calculer le temps de course du projectile, en résolvant l'équation $d_y = 0$. Il est alors possible de désigner une variable `t_max` qui calcule cette solution, ce qui permet de définir la limite du vecteur de temps `t_balle`.

```
t_max = (2 * v_0 * np.sin(a_0)) / g + t_0
t_balle = np.arange(t_0, t_max, 0.1)
```

Par la suite, on peut animer la trajectoire du projectile. Pour cela, c'est la `FuncAnimation` du module `matplotlib.animation` qui sera utilisée, car elle utilise une fonction pour animer les données calculées et non une boucle `for` comme utilise l'`ArtistAnimation`. Ce choix est justifié aussi par le fait que le programme sera mis sous la forme d'une classe par la suite.

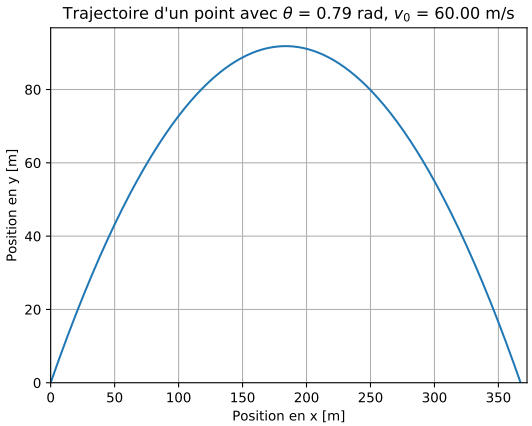


FIGURE 1 – Trajectoire fixe d'une balle

1.2 Ajouter les rebonds

Implémenter les rebonds, paraît assez simple et c'est le cas ! Il suffit de déterminer une nouvelle vitesse initiale à la fin du calcul de la trajectoire et d'en calculer une nouvelle, ce qui est très aisé avec une boucle `for` itérant pour le nombre de rebonds souhaités.

Ainsi, on met sous cette boucle la totalité de la section du code réservé au calcul de la trajectoire développé précédemment. A la fin de la boucle, la vitesse initiale est multipliée par un coefficient de restitution r, `v_0 = r*v_0` caractérisant la rigidité du sol, et dans l'ensemble, l'élasticité du rebond, c'est-à-dire la quantité d'énergie de la balle absorbée par le sol. La vitesse de la balle va donc diminuer au fur et à mesure des rebonds successifs. L'autre donnée à conserver lors du calcul du rebond est la position horizontal de la trajectoire précédente. On stocke donc la dernière position en x telle que : `d_0_x = d_x` et on ajoute `d_0_x` en offset au calcul de la position.

1.2.1 Exemple de bug rencontré : rebonds de la balle

Néanmoins, dans cette partie du développement de ce programme, des difficultés ont été rencontrées. D'après la figure 2, on voit que la forme de la trajectoire correspond vaguement à la figure recherchée, mais l'amplitude de la trajectoire sur l'axe y diminue très rapidement et la balle part dans les y négatifs.

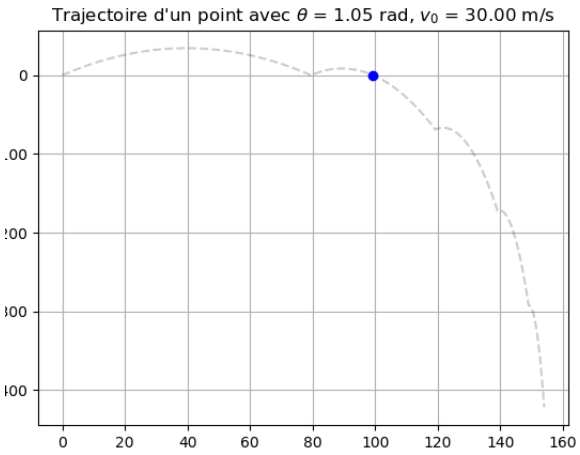


FIGURE 2 – Exemple de bug au moment de prendre en compte les rebonds pour la balle

Cependant, il est possible de remarquer très vite le problème : le temps de la trajectoire `t_max` est le même pour chaque rebond. En effet, cette variable était déterminé au début du programme et faisait partie des conditions initiales. Cependant, comme vu précédemment, `t_max` dépend de l'angle initiale, mais surtout de la vitesse initiale. Comme la vitesse varie après chaque rebond, il faut recalculer `t_max` après chaque rebond.

Notre balle a donc obtenu la capacité de rebondir n fois, mais elle se sent toujours très seule. Nous allons donc adapter le programme pour pouvoir en afficher plusieurs en simultanée.

1.3 Plus de balles

Dans cette partie, l'objectif est de pouvoir modéliser la trajectoire de **n** balles en simultané en définissant des constantes initiales aléatoires. Pour cela, il d'initialiser toutes ces conditions en leur donnant une valeur aléatoire avec le module `random`.

Afin de simplifier le programme dans son ensemble et l'implémentation de cette fonctionnalité, le code sera mis sous la forme d'une classe. La première version de ce programme est `class_multiball.py` : ce programme contient une classe "Balle" avec une méthode `setup` qui calcule la trajectoire de toutes les balles et une méthode `draw` qui les anime sur une figure. Cette syntaxe n'est cependant pas très cohérente et

```
class Balle_1():
    def setup(self):
        # calcule les trajectoires de n balles
    def draw(self):
        # trace toutes les balles

b = Balle_1()
b.setup()
b.draw()

#==========

class Balle_2():
    def __init__(self):
        # conditions initiales
    def get_tmax(self):
        # retourne le temps de course de la balle
    def change_speed(self):
        # fait diminuer la vitesse au moment du rebond
    def get_x(self, t):
        # retourne la position en x de la balle    t secondes
    def get_y(self, t):
        # retourne la position en y de la balle    t secondes
    def get_parab(self):
        # retourne une liste contenant les positions pour une premi re parabole
    def get_rebonds(self):
        # retourne la trajectoire compl te de la balle
    def get_traject(self):
        # adapte la liste de la m thode get_rebonds pour que toutes les
        # instances de la classe Balle_2 aient la m me longueur

b = Balle_2()
b.get_traject()
# Animation
```

FIGURE 3 – Comparaison des 2 versions du programme animant plusieurs projectiles

n'optimise pas les possibilités d'une classe. De plus, chaque méthode exécute un grand nombre d'action au lieu d'une seule.

Ainsi, une deuxième version a été créée : `class_multiball_2.py`. Ce programme contient une classe qui renvoie seulement une liste contenant les positions décrivant la trajectoire d'une balle. Le tracé est effectué à l'extérieur de la classe et fait bien retourner à la classe un objet qui est une unique balle.

Ce second programme m'a permis de bien comprendre le fonctionnement d'une classe : ici, la classe représente le "comportement" d'une balle. Néanmoins, l'animation est identique pour les 2 programmes, le seule changement étant la lisibilité et la logique derrière le calcul.

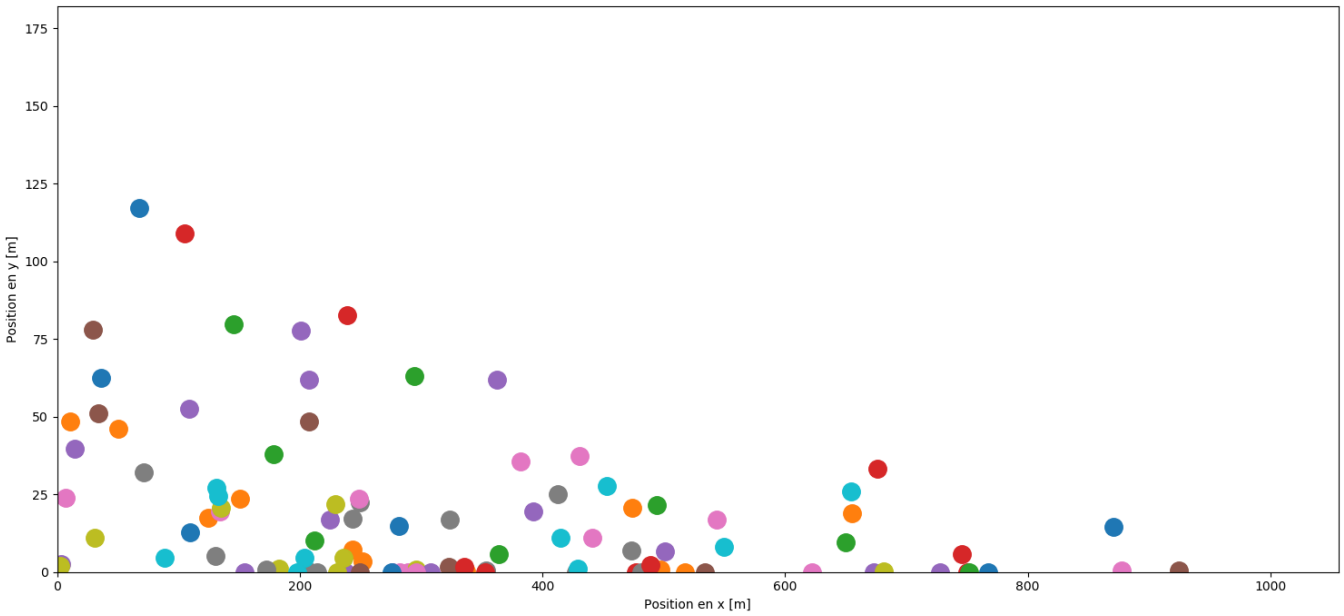


FIGURE 4 – Capture d'écran à un instant de la simulation pour 200 balles

Projet 2 Le tube de Kundt

Le tube de Kundt est un exemple de système où se propagent des ondes planes en 1D et a été étudié en cours d'Acoustique 1D pendant le semestre 3. L'objet de ce projet est de créer une animation simulant ce tube.

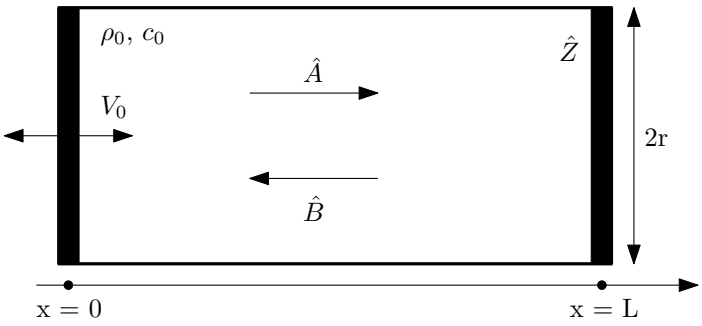


FIGURE 5 – Schématisation du tube de Kundt

2.1 Mise en équation du champ acoustique dans le tube

Le système considéré est donc un tube de Kundt de longueur L, cylindrique de diamètre $2r$. Les conditions aux frontières sont : en $x = 0$, un piston oscille à une vitesse $v_0 = V_0 \cos(\omega t)$ et en $x = L$, le tube est fermé par une paroi impédante d'impédance $\hat{Z}$. En considérant un régime monochromatique les amplitudes complexes de pression et de vitesse acoustique en jeu dans ce système sont :

$$\begin{cases} \hat{p}(x) = & \hat{A}\, e^{-ikx} + \hat{B}\, e^{ikx} \\ \hat{v}(x) = & \dfrac{1}{Z_c}\left(\hat{A}\, e^{-ikx} - \hat{B}\, e^{ikx}\right) \end{cases}$$

où $\hat{A}$ est l'amplitude de l'onde incidente, $\hat{B}$, l'amplitude de l'onde réfléchie et k, le nombre d'onde de l'onde sous réserve que $k = \dfrac{\omega}{c_0}$. $Z_c = \rho_0\, c_0$ est l'impédance caractéristique du milieu de propagation (ici de l'air donc $\rho = 1.2$ kg.m^{-3} et $c_0 = 340$ m.s^{-1}.

2.1.1 Conditions aux frontières

Les deux conditions limite dans le tube sont :

— en x = 0 :

$$\hat{v}(x = 0) = \frac{1}{Z_c}\left(\hat{A} - \hat{B}\right) = V_0$$

— en x = L :

$$\hat{Z}(x = L) = \frac{\hat{p}(x = L)}{\hat{v}(x = L)}$$

2.1.2 Champ acoustique de pression et de vitesse dans le tube

L'application de ces conditions aux frontière permet finalement d'écrire :

$$\begin{cases} \hat{p}(x) = & \hat{A}\left(e^{-ikx} + \hat{R}_p\, e^{ikx}\right) \\ \hat{v}(x) = & \dfrac{\hat{A}}{Z_c}\left(e^{-ikx} - \hat{R}_p\, e^{ikx}\right) \end{cases}$$

avec

$$\hat{R}_p = \frac{\eta - 1}{\eta + 1}\, e^{-2ikL} \quad \text{et} \quad \hat{A} = \frac{V_0 Z_c}{1 - \hat{R}_p}$$

2.2 Modélisation du champ acoustique de pression et de vitesse particulaire sans pertes dans le tube

En appliquant ces conditions, il est possible de simuler le champ acoustique dans le tube sans pertes. En observant l'animation, la distribution des champs de pression et de vitesses dans le tube de Kundt est visible et observable selon le temps. L'animation est réalisée en stockant l'évolution de chaque champ de pression (incident, réfléchi, total) et chaque champ de vitesse dans des variables séparées afin de pouvoir tracer l'ensemble dans deux sous-figures animées : la première représentant le champ de pression et la deuxième contenant les champs de vitesse. Dans ce cas, le champ acoustique est calculé dans un tube de $L = 10$ m, $r = 2$ m, $\eta = 0$, $\hat{R}_p = 0.7$ où le champ est monochromatique de fréquence $f = 520$ Hz, comme le montre la figure **??**.

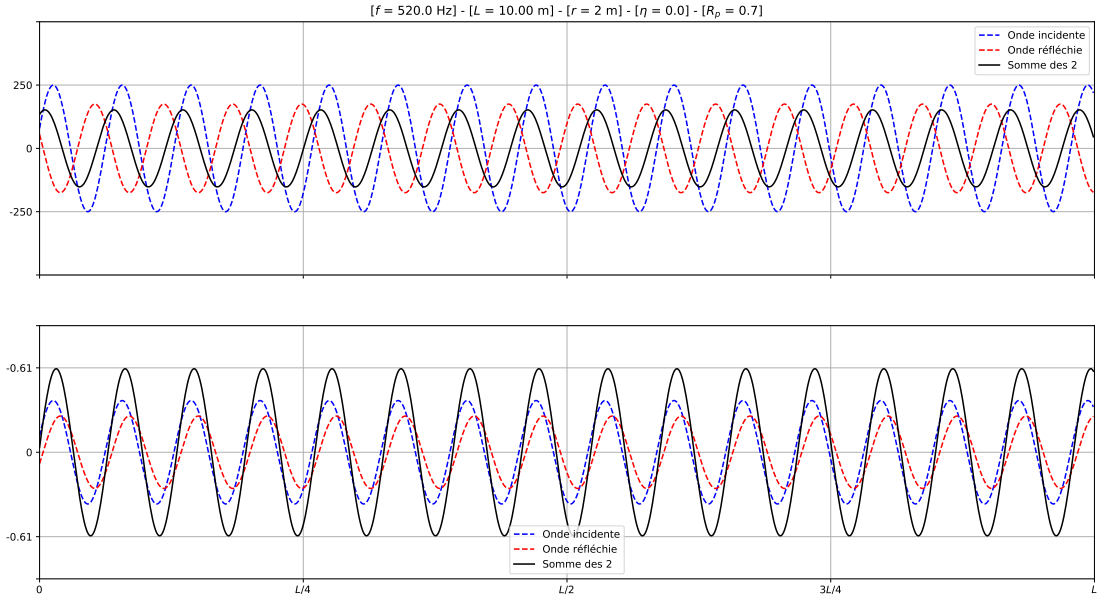


FIGURE 6 – Capture d'écran de la simulation du tube de Kundt sans pertes

2.3 Etude du coefficient de réflexion $\hat{R}_p$

Dans cette partie, le coefficient de réflexion $\hat{R}_p$ est à l'étude. Il a été établi dans l'étude théorique précédemment, l'expression du coefficient de réflexion : il dépend de η, un coefficient caractérisant l'absorption acoustique du matériau et de la fréquence f de l'onde. En calculant une matrice contenant les valeurs du coefficient de réflexion en fonction de η et de f, il est ensuite possible de tracer cette matrice dans un plan (η, f). Comme le coefficient de réflexion est complexe, le module et la phase sont calculés, ainsi que la partie réelle et imaginaire. Les résultats sont présentés en figure **??**.

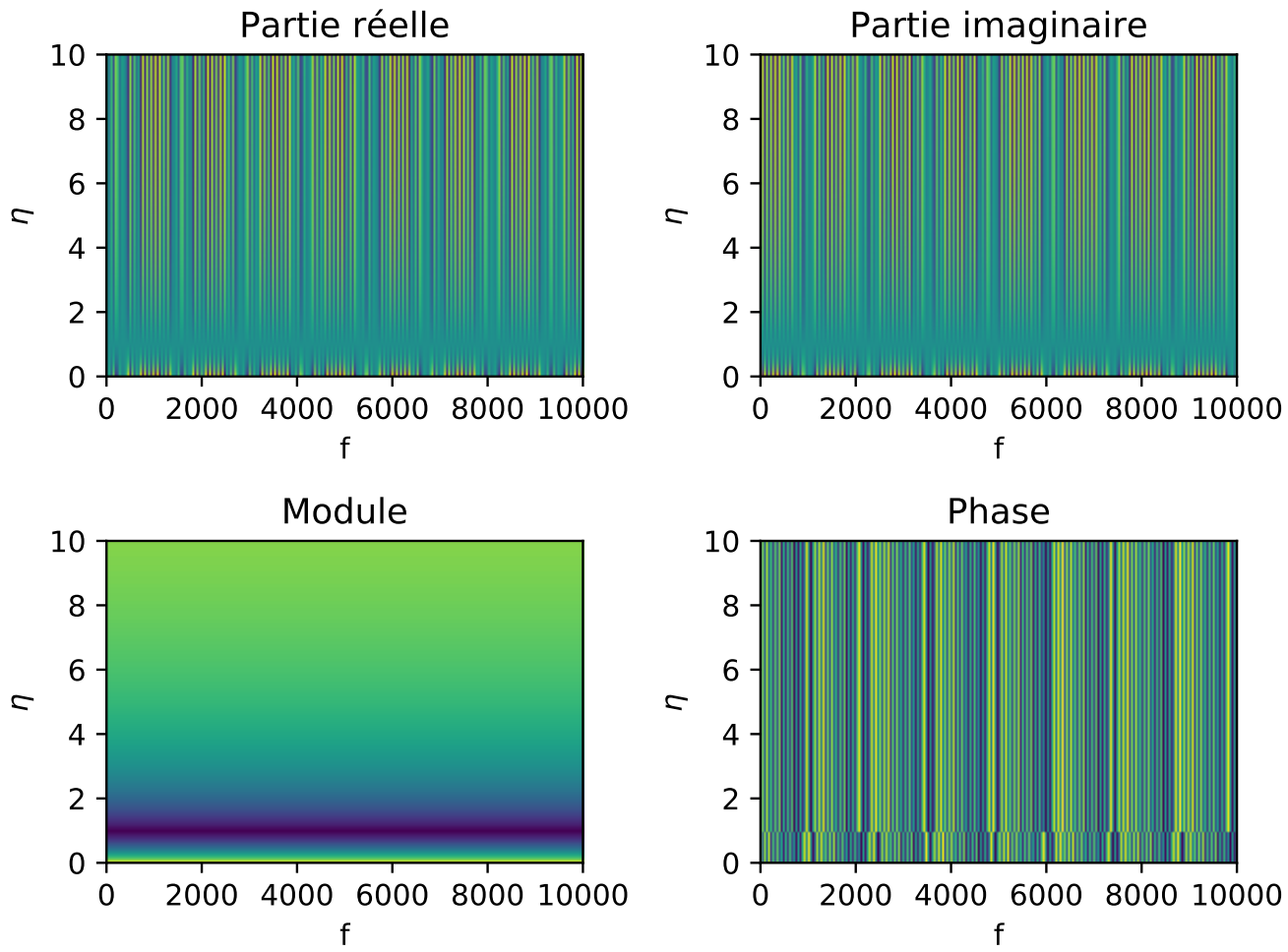


FIGURE 7 – Représentation des modules, phase, parties réelles et imaginaire du coefficient de réflexion dans un plan (η, f)

La partie réelle et imaginaire représentent une variation sinusoïdale du coefficient de réflexion en fonction de la fréquence : en effet, la fréquence se trouve dans une fonction exponentielle complexe dans l'expression de $\hat{R}_p$, qui, en réelle ou en imaginaire donne une fonction sinusoïdale. Selon η, il est possible de remarquer une discontinuité pour $\eta = 1$. Autour de cette valeur, le coefficient de réflexion diminue et tend vers 0. Il est cependant uniforme dans les $\eta > 1$. Cette discontinuité se retrouve dans le module et dans la phase : le module vaut 1 autour de $\eta = 0$, 0 pour $\eta = 1$, puis augmente pour les η supérieurs. Cette discontinuité montre que, quand le coefficient η d'un matériau est autour de 1, celui-ci est parfaitement absorbant car le coefficient de réflexion vaut 0. Le matériau est de plus en plus réfléchissant à mesure que η diminue si $\eta < 1$ et augmente si $\eta > 1$.

2.4 Simulation du champ acoustique dans le tube avec prise en compte des phénomènes de pertes

Dans cette dernière partie, la première partie comportant l'étude du champ acoustique est reprise et améliorée afin de prendre en compte les pertes. Les pertes acoustique dans un tube sont de 2 natures : dispersion et dissipation. Le résultat est identique pour les 2 phénomènes : une diminution de l'amplitude à l'extrémité opposée à la source dans le tube. Cependant, ce sont 2 phénomènes physique différents : la dissipation est la perte d'énergie d'une onde avec la distance et la dispersion est la variation de la célérité en fonction de la fréquence de l'onde.

Ainsi, le nombre d'onde acoustique peut être réécrit :

$$k_d = \frac{\omega}{c} + (1 - i)\alpha \tag{1}$$

avec α, le coefficient d'absorption :

$$\alpha = 3 \cdot 10^{-5}\sqrt{f}/r \tag{2}$$

Une capture d'écran du programme avec le nouveau nombre d'onde implémenté est présentée en figure **??**.

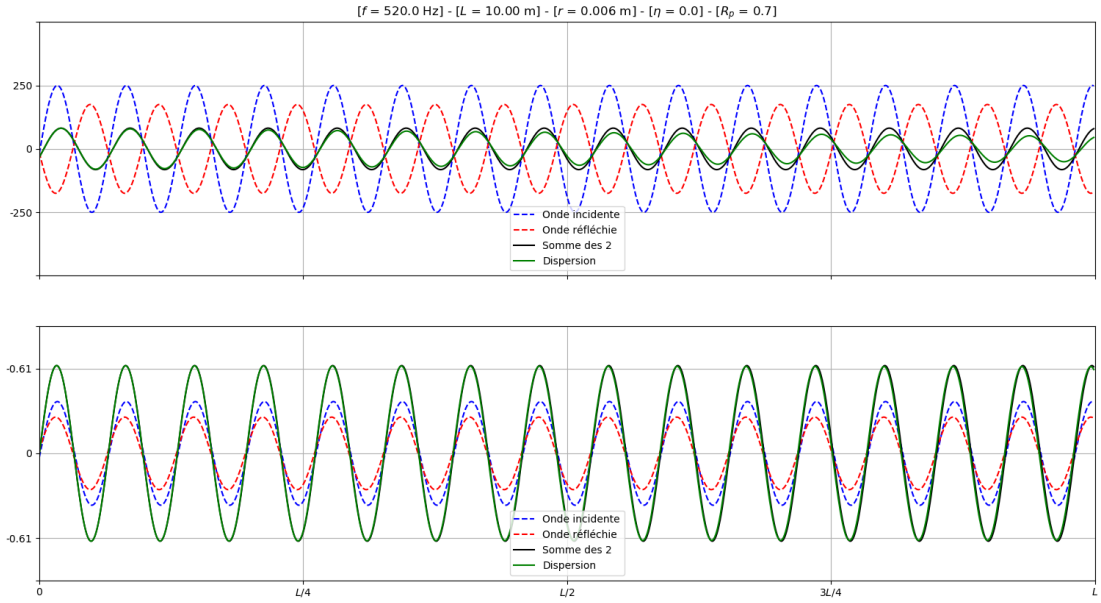
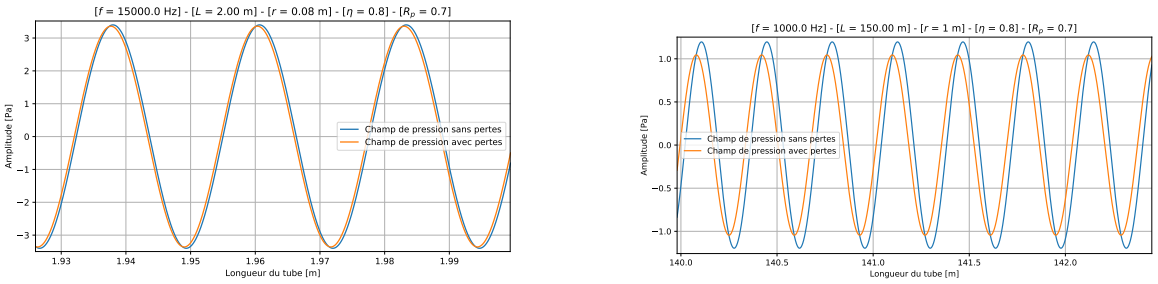


FIGURE 8 – Capture d'écran de la simulation du champ acoustique dans le tube en prenant en compte les phénomènes de pertes

Cette expression théorique montre, qu'en théorie, il y aura de la dispersion à une fréquence élevée et un rayon faible de la section du tube, soit rf. La dissipation aura lieu, quant à elle, quand la longueur du tube est grande et s'il y a des ondes stationnaires : plus l'onde stationnaire se situe sur un minimum de pression, plus l'onde est atténuée. Il est possible de visualiser le phénomène de dispersion sans l'atténuation dans un tube assez court, mais l'atténuation s'accompagne généralement de dispersion.



(a) Dispersion dans un tube (b) Atténuation et dispersion du champ acoustique

FIGURE 9 – Phénomènes de pertes dans différentes configurations pour un tube de Kundt

En observant la figure **??**, il est possible de voir, dans un tube court (2m ici) le phénomène de dispersion : l'onde en orange semble déphasée par rapport à l'onde. En fait, sa fréquence est plus faible, du fait de la variation du nombre d'onde avec la fréquence et le rayon du tube : c'est le phénomène de dispersion. Pour observer une atténuation, c'est-à-dire une perte d'énergie, il faut allonger le tube de Kundt : dans la figure **??**, le tube mesure 150m et on visualise le champ acoustique avec et sans pertes au bout du tube. Il est possible d'observer une nette diminution de l'amplitude ainsi qu'une forte dispersion : il est impossible de visualiser une atténuation sans dispersion dans ce cas, en utilisant la variation de la célérité de l'onde avec la fréquence.

La simulation numérique du tube de Kundt a permis de mettre en application des connaissances théorique (cours d'acoustique 1D du semestre 3) afin de comprendre visuellement l'évolution du champ acoustique dans ce tube, approfondir le phénomène de pertes et de comprendre de quels facteurs dépend de le coefficient de réflexion.

Projet 3 L'interface du reste de ma vie

Pour ce troisième projet, nous allons créer une interface permettant de charger un fichier généré par le logiciel *INTAC* du CTTM et d'afficher les résultats. Pour cela, il s'agit de décomposer le travail en plusieurs étapes : créer les fonctions pour charger un fichier texte, puis le tracer, et ensuite créer une interface utilisateur avec des boutons et un espace d'affichage afin de pouvoir visualiser la courbe et interagir avec pour modifier par exemple l'affichage ou le type de tracé (temporel ou fréquentiel).

La première étape est assez simple : demander à l'utilisateur le chemin du fichier à charger et le tracer demande seulement quelques lignes en utilisant l'instruction `data = np.loadtxt(filename)` où `filename` est une chaîne de caractères contenant le chemin du fichier texte à charger. Ensuite, il suffit de créer une figure avec matplotlib pour la tracer.

La seconde et principale étape est plus délicate : il faut utiliser un module permettant de créer une interface graphique pour qui utilise ces fonctions. Il existe de nombreux modules qui permettent de créer des GUI (graphical user interface) tels que Tkinter, PyQt, Kivy ou beaucoup d'autres. Mon choix s'est porté sur le module PyQt pour sa simplicité à utiliser avec des classes, et pour le fait qu'il permette de faire plus simplement une belle interface que Tkinter, qui est assez brut. La difficulté de ce module reste tout de même que la documentation est en C et non en python, mais de nombreux exemples et tutoriels existent pour prendre en main ce module. Enfin, j'ai choisi ce module car il comprend une intégration plutôt simple de Matplotlib, que ce soit pour la figure ou la barre d'outils.

3.1 Créer l'interface

Le programme de l'interface est donc construit sous la forme d'une classe "GUI", initialisée avec des variables précisant la géométrie de la fenêtre, son titre, icône et des variables utiles au fonctionnement des différentes fonctions de l'interface. De façon générale, l'initialisation de cette classe doit comprendre tous les objets que l'on veut afficher dans l'interface. La création des objets et la disposition de ces derniers sont donc séparés dans 2 méthodes différentes appelées dans l'initialisation. Cette classe est de type `QDialog`. Il y a plusieurs types de classe pour créer une interface dont : `QWidget`, `QDialog`, `QMainWindow`. On utilise l'une ou l'autre de ces classes selon l'interface à créer :

- `QMainWindow` permet la création d'une barre de manu, d'une barre d'outils et d'autres objets dont une fenêtre principal peut avoir besoin.
- `QDialog` est plus utile pour une petite interface. Elle est basée sur `QWidget` et est designée pour apparaître comme une fenêtre. De plus, elle a des fonctions qui rendent la création de boutons assez basique simple et bien intégrée à la fenêtre.
- Enfin `QWidget` est la classe de base pour tout objet affichable sous Qt. Quand un objet widget n'a pas de fenêtre parent, il peut être affiché comme une fenêtre.

Le choix d'utiliser `QDialog` est relativement arbitraire mais est justifié par le fait que pour le développement de cette interface, il nous faut avoir la possibilité de faire afficher une fenêtre "enfant" pour afficher l'arborescence du disque et choisir un fichier à tracer au lieu de devoir chercher le chemin de chaque fichier, c'est l'objectif premier de cette interface. Il n'est pas possible d'ouvrir une boîte de dialogue avec `QWidget`. Enfin, `QMainWindow` paraît un peu exagéré : une barre d'outils/ de menus est inutile pour une interface de cette taille.

3.1.1 Disposer les éléments dans une fenêtre

Pour commencer et avant de travailler l'algorithmique derrière l'interface, il s'agit de créer les boutons et les différents espaces de la fenêtre. Pour cela, PyQt possède une fonctionnalité intéressante : les layouts. Les layouts sont des espaces pouvant contenir des objets comme des boutons, un menu déroulant, du texte ou des cases à cocher, permettant d'utiliser un placement relatif pour tous ces objets plutôt qu'une disposition absolue (où chaque bouton est placé selon des coordonnées précises sur la fenêtre). Pour créer la fenêtre initiale, il y a 2 méthodes : la première est de tout faire en ligne de code et à la main. Chaque bouton

est ainsi créé, paramétré et placé manuellement dans son layout. Cette méthode est assez longue et rend le programme de l'interface très long et peu lisible.

Cependant, c'est la seule méthode que je connaissais quand j'ai crée la première version de ce projet, avant de découvrir le logiciel *Qt Designer* qui permet de disposer les layouts et les boutons visuellement. Il suffit ensuite d'importer le fichier que sort le logiciel dans le script et de connecter les boutons aux méthodes associées. Cette seconde méthode est beaucoup rapide et permet de faire des interfaces bien plus esthétiques. De plus, en plaçant correctement les boutons dans un *layout*, la fenêtre s'adapte en cas de redimensionnement.

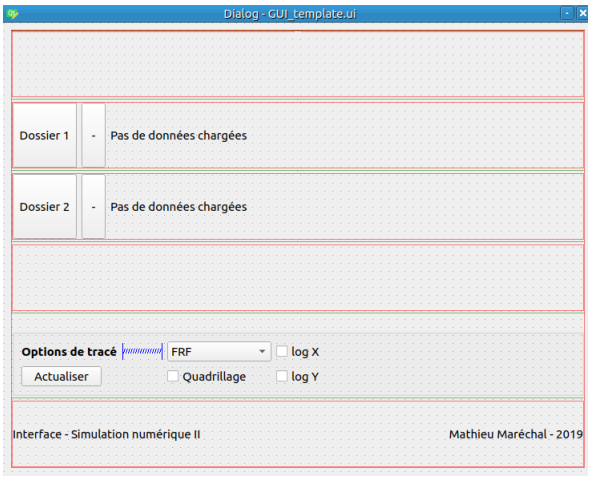


FIGURE 10 – Capture d'écran de l'interface du projet 3 réalisée avec Qt Designer

3.1.2 Chargement de fichiers et tracé de données

Maintenant que l'interface a tous les boutons nécessaires, il faut les connecter à des méthodes. Pour cela, on utilise `button.clicked.connect(methode)`. Cette ligne permet de lier un bouton à une méthode, qui sera appelée au moment du clic sur le bouton "button". Pour commencer, il faut avoir des fichiers à tracer : on peut, par exemple, ouvrir une boîte de dialogue permettant à l'utilisateur de sélectionner les données à tracer. Les méthodes disponibles avec ce module permettent de proposer de charger un fichier ou un ensemble de fichier avec la méthode `QFileDialog.getOpenFileName()` ou de charger un dossier entier `QFileDialog.getExistingDirectory()`. La première version de l'interface permettait de charger un fichier à la fois, mais la version actuelle prend en charge un dossier complet avec plusieurs types de tracés possible.

Les méthodes ouvrant une boîte de dialogue retourne le chemin absolu du fichier sous forme d'une chaîne de caractères. Une fois que l'on stocke ce chemin dans une variable, on peut l'appeler dans la méthode `np.loadtxt` de numpy pour charger les données d'un fichier précis. En effet, les dossiers générés par le logiciel du CTTM à l'exportation des mesures comportent des fichiers avec des noms toujours identique :

— pour une acquisition temporelle, `TemporalData.txt`
— pour une FRF, `FRF_ModPhase.txt`
— pour un spectre, `PowerSpetrum.txt`

En connaissant les noms de chaque type de fichier, il est possible de créer un menu déroulant détaillant le type de fichiers que l'utilisateur souhaite visualiser et d'importer un fichier précis selon ce choix. Le type de tracé et l'import des fichiers est détaillé dans la méthode `loadData` de la classe de l'interface. Après l'appel de cette fonction, les fichiers sont importés, il ne reste plus qu'à tracer les courbes !

La difficulté d'avoir la possibilité de tracer 2 jeux de données en même temps est que, quand l'utilisateur ne veut observer qu'une seule mesure, il faut prendre en compte que seul le dossier 1 ou le dossier 2 contient des données à tracer, et ainsi gérer les erreurs en utilisant des opérateurs booléens renvoyant l'état des différentes étapes comme `loaded_1` et `loaded_2` qui déterminent respectivement si le dossier 1 ou le dossier 2 contient des données à charger. En ajoutant la gestion des erreurs avec les structures `try: except`, le programme s'adapte à toutes les conditions possible : selon l'erreur, le programme renvoie un message

d'erreur, c'est-à-dire une fenêtre qui s'ouvre et qui précise la source de l'erreur. L'essentiel de la gestion d'erreur est de faire en sorte que l'interface ne *crash* pas.

Le résultat final est présenté dans une capture d'écran en figure **??**. Des réponses en fréquences de l'excitation d'une plaque en plexiglas par un pot vibrant en 2 points différents sont chargés pour cet exemple.

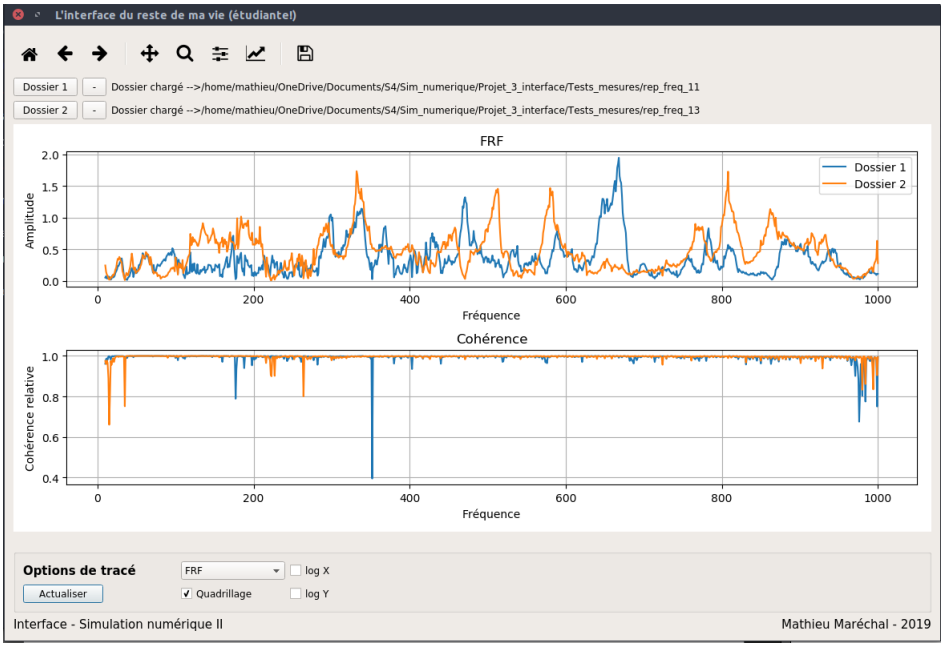


FIGURE 11 – Capture d'écran de l'interface final du projet 3

3.2 Mode d'emploi de l'interface

Le but de ce programme est de tracer les mesures issues d'acquisitions du logiciel *Analyseur CTTM* exclusivement. Pour commencer, il faut cliquer sur le bouton "Dossier 1" ou "Dossier 2" situé en haut de l'interface. Une boîte de dialogue s'ouvre alors et demande de sélectionner un dossier. Une fois sélectionné, le chemin du fichier s'affiche à droite du bouton. Il faut maintenant sélectionner le mode de tracé souhaité dans la zone "Options de tracé" et ensuite cliquer sur le bouton "Actualiser". Les données peuvent être déchargées du programme en cliquant sur le bouton "-" situé à droite du bouton de chargement. Jusqu'à 2 jeux de données peuvent être chargés à la fois. Finalement, les fonctionnalités proposées par les figures matplotlib classiques sont disponibles ici : zoom, sauvegarde des figures, ...

Projet 4 Traitement du signal et imagerie

Ce quatrième projet sur le sujet du traitement de signal : l'objectif est de tracer et traiter de différentes manières un fichier de données puis de traiter un grand nombre de mesure dans une seconde partie.

4.1 Traitement du signal acoustique picoseconde

L'objectif de cette partie est d'apprendre la démarche de réflexion et de traitement à avoir pour extraire des informations d'un signal, quel qu'il soit. Tout d'abord, il s'agit de charger le fichier et de le tracer. Ensuite, il faut calculer le module du spectre (en suivant la méthode détaillée dans le sujet du projet) avec les fonctions `np.fft.ftt` et `np.fft.fftfreq` du module numpy, afin de visualiser le contenu fréquentiel du fichier `time_rawsignals_84GPa_nice.txt`. En prenant en compte le fait que l'axe temporel est gradué en nanosecondes (ns), l'unité de l'axe fréquentiel est en GHz, soit 10^9 Hz.

Projet 5 Optique géométrique

Pour ce cinquième et dernier projet, il est demandé de mettre en évidence le stigmatisme de différents systèmes optiques. il s'agit de partir de notions d'optique géométrique pour démontrer les conditions menant à un stigmatisme approché et un système non stigmatique. Un système optique est dit stigmatique quand les rayons sortants de ce système convergent en un point précis (ou un ensemble de points très localisée dans le cadre d'un stigmatisme approché (le stigmatisme rigoureux n'existe que théoriquement, pour la réflexion d'un objet sur un miroir plan). Si le système est non-stigmatique, la convergence des rayons en sortie formera une tâche.

L'objectif de ce projet est donc de mettre en évidence ce phénomène pour : les miroirs sphérique concave et convexe et les lentilles sphériques biconcave et biconvexe. Les dioptres seront caractérisés par leur sommet (noté a), leur rayon, noté r ainsi que leur diamètre d'ouverture ou épaisseur, notée e.

Par la suite, en réfléchissant aux différentes étapes du problème, j'ai réalisé que la difficulté de ce projet n'est pas de nature algorithmique mais géométrique. En effet, pour calculer la trajectoire de rayons lumineux à travers ces lentilles et miroirs, il s'agit d'abord d'établir des relations géométriques donnant la trajectoire de ces rayons. En implémentant ces trajectoires, il ne reste plus qu'à tracer les systèmes optiques avec les rayons associés.

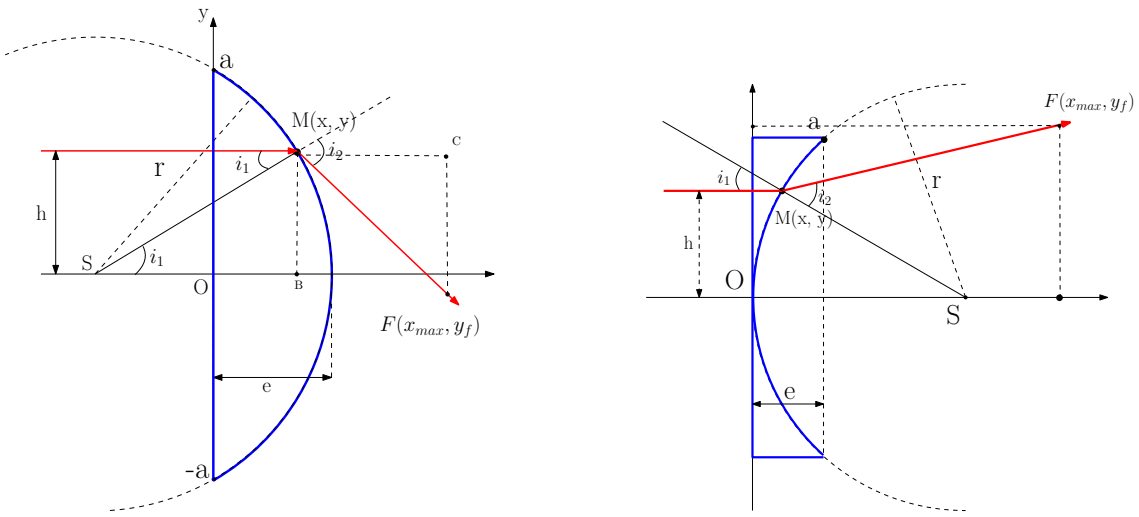
5.1 Forme des dioptres

Tout d'abord, il s'agit de tracer les différents éléments optique. Tous les éléments pour lesquels les conditions de stigmatisme sont à illustrer sont constitués d'un ou plusieurs dioptre, qui est soit concave, soit convexe. Pour déterminer toutes les relations mathématiques et géométrique nécessaire au calcul de la trajectoire des rayons, des schémas sont présentés en figure 12. Il est possible de réaliser les calculs de trajectoires pour ces 2 objets seulement et d'en déduire tous les autres à partir des relations mathématiques et géométrique établies pour ces 2 lentilles. Pour cela, des schémas sont présentés en figure 12.

Tout d'abord, il s'agit d'établir l'équation décrivant la portion de cercle du dioptre en question. L'équation est donc, à partir de l'équation cartésienne du cercle,

$$x(y) = \sqrt{r^2 - y^2} - r + e \tag{3}$$

puisque l'origine du repère considéré est placé au niveau du point minimum du dioptre (sur l'axe horizontal), la portion de cercle est décalée de $-r + e$. Pour obtenir l'arc de cercle (en bleu sur les schémas), y est un vecteur qui varie de $-a$ à a.



(a) Dioptre convergent (convexe) (b) Dioptre divergent (concave)

FIGURE 12 – Schémas et notations pour les 2 types de dioptres, ici des lentilles

5.2 Trajectoire des rayons pour une lentille convergente

Pour connaître la trajectoire des rayons dans ce cas, le schéma de la figure **??** est utilisé. Le schéma permet de remarquer que seulement 3 points sont nécessaire pour tracer un rayon : le point d'origine (considéré à l'infini), le point d'intersection sur le dioptre et le point à l'infini vers les x croissants. Dans le programme, deux listes seront alors déclarées :

```
x = [-lim, x_m, lim]
y = [h, h, y_f]
```

avec h, la hauteur initiale du rayon et `lim`, la limite du domaine. On tracera par la suite les rayons entre $x_{min} = -200$ et $x_{max} = 200$, donc `lim = 200`. Il faut donc maintenant calculer x_m et y_f.

En considérant le triangle SBM, il est possible d'établir une relation afin de déterminer x_m. Pour cela, le point B est posé et a pour coordonnées $(x_m, 0)$, c'est-à-dire que OB $= x_m$. Il apparaît que SB $= r \cos i_1$. Or, SB $= r - e +$ OB. Finalement,

$$x_m = r\cos(i_1) - r + e$$

Pour calculer la hauteur du point F, on pose un point C représenté sur le schéma, et on considère le triangle MCF. l'angle en M vaut ici $i_2 - i_1$. Il vient donc que : $\tan(i_2 - i_1) = \frac{CF}{MC}$. Or, MC $= x_{max} - x_m$ et CF $= y_f - h$. Finalement, il vient que :

$$y_f = h - (x_{max} - x_m)\tan(i_2 - i_1)$$

Enfin, d'après la loi de Snell-Descartes, $\sin(i_1) = n\sin(i_2)$, avec n, l'indice de réfraction du matériau. Or, $i_1 = \arcsin(\frac{h}{r})$, donc $i_2 = \arcsin(n\frac{h}{r})$

5.3 Trajectoire des rayons pour une lentille divergente

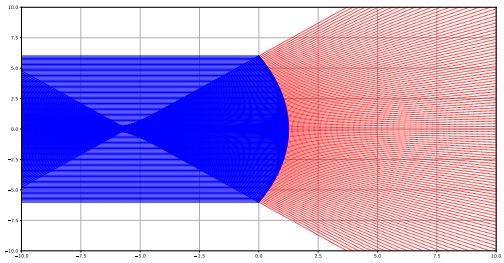
Pour une lentille divergente, les calculs sont identiques mais comme le rayon réel diverge de l'axe optique, y_f s'exprime comme étant $y_f = h + (x_{max} - x_m)\tan(i_2 - i_1)$, ce qui est logique avec la représentation visuelle. L'autre point à exprimer dans ce cas, est la trajectoire du rayon virtuel réfléchi. Cependant ce rayon a la trajectoire inverse du rayon divergent réel, donc le tracé ne pose pas de problème : on pourra déclarer la hauteur du rayon virtuel réfléchi comme `y_refl = -y_f`.

5.4 Simulation de la trajectoire de rayons lumineux dans un système optique

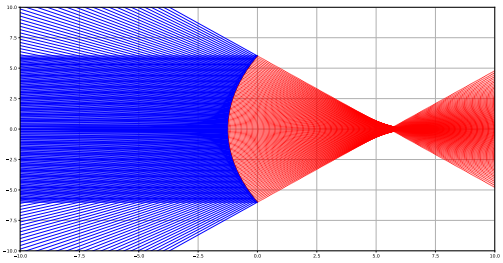
Pour cette dernière étape, il reste à créer le programme. L'idée est de créer 2 classes, une pour les miroirs et l'autres pour les lentilles.

5.4.1 Miroirs sphériques concave et convexe

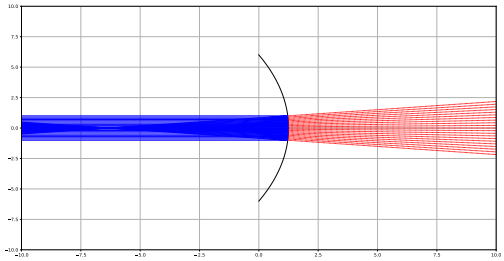
La classe `Miroir` est très simple : les variables d'initialisation sont le rayon du miroir, son sommet, son épaisseur ainsi que l'équation du miroir sphérique. Ensuite, une méthode `concave` retourne une liste contenant l'équation x en fonction du vecteur y et la méthode `convexe` retourne l'équation opposée : le dioptre divergent est symétrique au dioptre convergent par rapport à l'axe $y = 0$, ce qui sera aussi utile au moment de créer une lentille biconvexe. Après avoir pris en compte cette opération, il faut noter que le point x_m désignant le point d'impact du rayon sur la surface du dioptre sera opposé lui aussi.



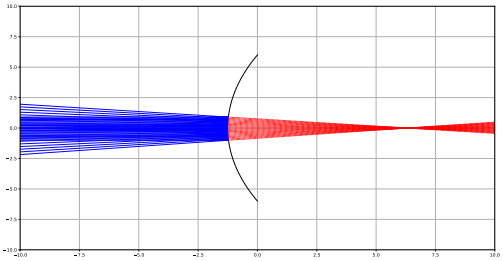
(a) Miroir concave avec faisceau large



(b) Miroir convexe avec faisceau large



(c) Miroir concave avec faisceau fin



(d) Miroir convexe avec faisceau fin

FIGURE 13 – Miroirs sphériques concave et convexe dans les cas où le faisceau lumineux est large ou fin

Les figures n'étant pas suffisamment visibles dans ce format, elles sont aussi présentées en annexe et peuvent être affichés en compilant les scripts python dans le dossier du projet 5. Il apparaît ici que les dioptres en question (ici des miroirs) peuvent être considérés comme des systèmes stigmatiques approchés quand les rayons lumineux sont situés vers le centre du miroir sphérique. Pour la hauteur du faisceau, le rayon de la lentille est très élevé (on peut même dire que $r \longrightarrow \infty$) donc le miroir sphérique se comporte comme un miroir plan, qui est un système stigmatique. Or, d'après le cours d'optique géométrique de L1 SPI, si la taille du faisceau est négligeable devant le rayon du dioptre, alors le système est dit "dans les conditions de Gauss". On peut donc en conclure à l'aide de ces illustrations qu'un système optique peut être considéré comme stigmatique dans les conditions de Gauss, c'est-à-dire quand le faisceau est localisé au centre du système optique et pas sur les bords.

5.4.2 Lentilles sphériques biconcaves et biconvexes

La seconde classe à développer dans ce projet est la classe `Lentille` comprenant des méthodes afin de tracer les différents types de lentilles : `biconcave`, `biconvexe` et `convexe` (pour tester les formules trigonométrique établies précédemment) ainsi qu'une méthode `rayon` qui calcule les trajectoires pour chaque cas.

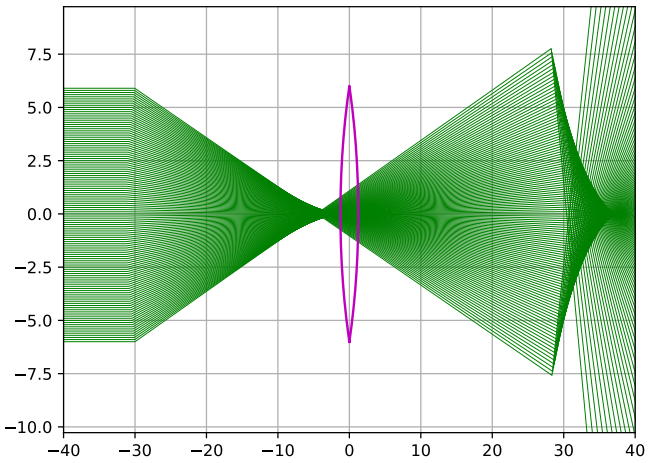


FIGURE 14 – Bug lentille

Table des figures

Annexes

A Représentation de la trajectoire des rayons dans les différents systèmes optiques