Énoncés des projets de Simulation Numérique II

(*Nom de code* – Dém***ez-vous !)

Philosophie du semestre et évaluation

La philosophie du semestre réside dans le nom de code des projets : « **Dém***ez-vous ! ». Après le premier semestre plutôt orienté vers l'apprentissage d'outils avancés, le but de ce semestre est que vous développiez vos capacités à **mettre en place par vous-même une stratégie de développement pour répondre à un cahier des charges, à une demande, à une idée de programme informatique**. Nous vous demanderons pour cela un **travail individuel** et nous évaluerons aussi bien le résultat final de votre travail (les codes) que votre réflexion et votre parcours pour y arriver. Une fois n'est pas coutume en « informatique », on va donc vous demander de nous rendre un rapport, en plus des codes (évidemment !). Ce rapport devra relater/expliquer votre démarche, les difficultés rencontrées et les solutions trouvées/retenues, vos choix algorithmiques, etc., pour chaque projet. **Pensez donc à prendre des notes !** On attend de vous que vous traitiez l'ensemble des projets. Bien entendu, **les séances de TP sont obligatoires** et seront des moments privilégiés pour que vous puissiez échanger avec nous, poser vos questions, demander des conseils, ...

Pour pimenter le tout, nous organisons un **concours BONUS** dont les gagnants se verront récompenser d'un bonus (2 pts) sur leur note finale. Pour faire partie des heureux lauréats (au nombre de 6), il faudra nous rendre :

- la plus belle interface,
- le code le plus élégant,
- le code le plus compact,
- le code le mieux documenté,
- le code le plus rapide,
- le code le plus fonctionnel (au sens code écrit avec des classes, fonctions, et/ou modules).

S'il s'avère qu'un (ou plusieurs) étudiant(s) est(sont) classé(s) premier(s) sur plusieurs catégories, un bonus supérieur lui(leur) sera accordé et les seconds feront alors partie des lauréats.

Dans ce monde d'équilibre, nous proposons également un **concours MALUS** dont les gagnants se verront récompenser d'un malus (-10 pts par succès) sur leur note finale. Pour faire partie des heureux lauréats (au nombre indéfini), il faudra nous rendre :

- le code le plus semblable à celui d'un ou plusieurs autres camarades.

Ici pas de jaloux, tout le monde peut gagner et même plusieurs fois !

Projet 1 : La balle

1.1 Trajectoire d'un projectile

On considère un objet lancé avec une vitesse initiale $\boldsymbol{v_0}$ dans une direction faisant un angle θ avec l'axe des abscisses. L'objet est lancé par une vitesse initiale $\boldsymbol{v}_0$ au point de coordonnées $(x = 0, y = 0)$.

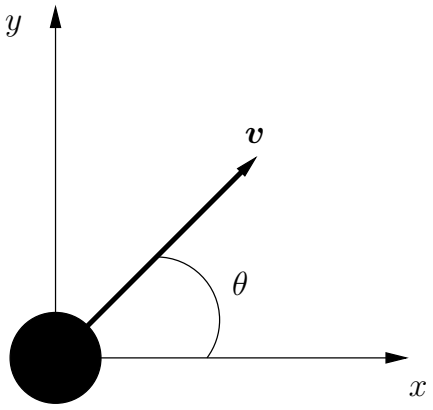

FIGURE 1 – Vue schématique du projectile dans son repère.

En considérant le schéma de la figure 1, le bilan des forces appliqué au projectile s'écrit :

$$\sum \boldsymbol{F}_{ext} = m\boldsymbol{a} \Longleftrightarrow \boldsymbol{g} = \boldsymbol{a},$$

où $\boldsymbol{g}$ est la gravité et $\boldsymbol{a}$ l'accélération. Les composantes du vecteur accélération s'écrivent alors :

$$\begin{cases} a_x &= 0 \\ a_y &= -g \end{cases}$$

L'intégration de l'accélération par rapport au temps t permet d'obtenir les composantes (v_x, v_y) de la vitesse $\boldsymbol{v}$ du projectile :

$$\begin{cases} v_x &= v_{0_x} \\ v_y &= -gt + v_{0_y} \end{cases}$$

où v_{0_x} et v_{0_y} sont des constantes d'intégration. A l'instant $t = 0$ la vitesse $\boldsymbol{v}$ du projectile est égale à la vitesse initiale $\boldsymbol{v}_0$. Cette condition initiale permet de déterminer les constantes d'intégration v_{0_x} et v_{0_y} et de réécrire les composantes de la vitesse $\boldsymbol{v}$ sous la forme :

$$\begin{cases} v_x &= v_0 cos(\theta) \\ v_y &= -gt + v_0 sin(\theta) \end{cases}$$

La position $\boldsymbol{d}$ du projectile est alors déterminée en intégrant sa vitesse par rapport au temps t :

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

où d_{0_x} et d_{0_y} sont des constantes d'intégration. À l'instant $t = 0$, le projectile est localisé en $x = 0$ et $y = 0$ donc :

$$\boxed{\begin{cases} d_x &= v_0 t cos(\theta) \\ d_y &= v_0 t sin(\theta) - \frac{1}{2}gt^2 \end{cases}}$$

Une fois lancé, le projectile touche alors le sol (situé en $y = 0$) lorsque :

$$t\left(v_0 sin(\theta) - \frac{1}{2}gt\right) = 0.$$

Cette équation possède deux solutions :

$$\boxed{\left\{\begin{array}{ccccc} t & = & t_0 & = & 0 \\ t & = & t_{max} & = & 2v_0 sin(\theta)/g \end{array}\right.}$$

La première solution (t_0) correspond au temps initial, pour lequel le projectile a une altitude nulle ($d_y = 0$). La seconde solution (t_{max}) est le temps écoulé entre le lancé et la collision avec le sol. À cet instant le projectile a à nouveau une altitude nulle ($d_y = 0$). La trajectoire du projectile devra donc être calculé entre le temps $t = 0$ et le temps $t = t_{max}$.

1.2 Rebond d'un projectile

Afin d'améliorer le modèle physique présenté précédemment, il est possible de prendre en compte les rebonds du projectile. La collision entre deux corps est généralement caractérisée par un coefficient traduisant l'élasticité du rebond. Ce coefficient est le ***coefficient de restitution*** défini tel que :

$$r = \frac{\boldsymbol{v}_r}{\boldsymbol{v}_c},$$

où $\boldsymbol{v}_r$ est la vitesse du projectile au rebond et $\boldsymbol{v}_c$ est la vitesse du projectile au moment de la collision comme schématisé sur la figure 2.

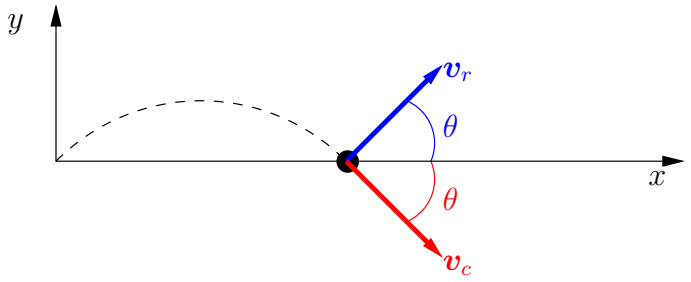

FIGURE 2 – Vue schématique du projectile dans son repère.

Ce coefficient de restitution est compris entre 0 et 1 [1]. À titre d'exemple, le tableau 1 regroupe quelques coefficients de restitution.

Projectile	**Surface**	r
Boule en acier	Sol en acier	0.95
Balle de tennis	Terre battue	0.85
Balle de tennis	Sol dur	0.71
Balle en bois	Sol en bois	0.5

TABLE 1 – Exemple de coefficients de restitution

1. Un coefficient de restitution supérieur à 1 est théoriquement impossible : il traduirait une production d'énergie !

Après le lancé, le premier rebond d'un projectile s'effectue au temps $t = t_{max}$ avec une vitesse de rebond égale à $r\boldsymbol{v}_c$. La vitesse du projectile à l'impact, en $t = t_{max}$, peut s'écrire :

$$\left\{ \begin{array}{rcl} v_x & = & v_0 cos(\theta) \\ v_y & = & -gt_{max} + v_0 sin(\theta) \quad = \quad -v_0 sin(\theta) \end{array} \right.$$

Les composantes de la vitesse de rebond sont alors :

$$\left\{ \begin{array}{rcl} v_{0_{rx}} & = & rv_0 cos(\theta) \\ v_{0_{ry}} & = & rv_0 sin(\theta) \end{array} \right.$$

Optionnel : Une autre amélioration du modèle consiste à prendre en compte le temps de collision. Celui-ci influe sur l'accélération au rebond comme suit :

$$a = \frac{v_{rebond} + v_{impact}}{\tau}$$

avec τ le temps de collision.

1.3 Travail à réaliser

- Écrire un code permettant de calculer et animer la trajectoire d'un projectile (Cf. exemple de la vidéo ***ball.mp4***)
- Améliorer ce code afin de prendre en compte les rebonds du projectile. Votre code permettra de choisir le nombre de rebond.
- Écrire un code permettant d'animer plusieurs projectiles lancés à des angles, des vitesses initiales et des décalages temporels aléatoires (Cf. exemple de la vidéo ***multiball.mp4***)
- ***Optionnel :*** Et pourquoi n'y aurait il pas d'obstacle ? Modifier votre code pour prendre en compte la présence d'un éventuel mur comme présenté en figure 3.

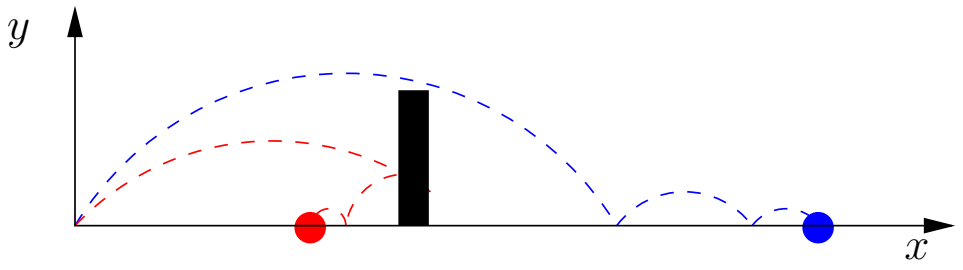

FIGURE 3 – Vue schématique du projectile en présence d'un mur.

Projet 2 : Le tube de Kundt

2.1 Aspects Théoriques

2.1.1 Champ acoustique

Le système considéré, présenté à la figure 4, est un tube de Kundt cylindrique de longueur L et de rayon r terminé en $x = L$ par une paroi d'impédance $\hat{Z}$. En $x = 0$, un piston oscille à la vitesse $v_0 = V_0 cos(\omega t)$ avec $\omega = 2\pi f$ la pulsation, f la fréquence et t le temps.

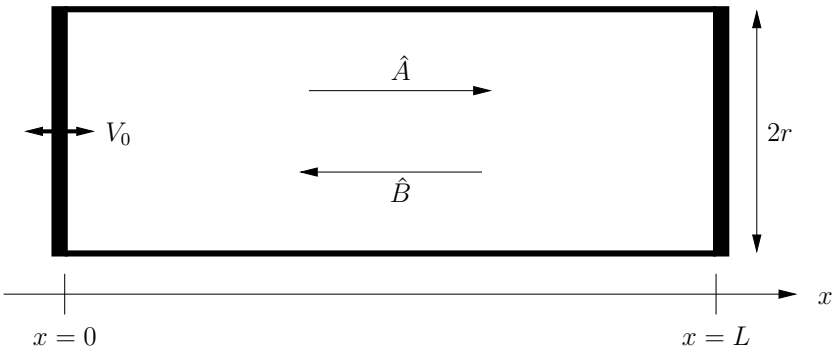

FIGURE 4 – Représentation schématique du tube de Kundt.

En régime monochromatique, les amplitudes complexes de pression et de vitesse acoustique dans ce système peuvent s'écrire comme suit :

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

où $\hat{A}$ et $\hat{B}$ sont respectivement les amplitudes de l'onde incidente et celle de l'onde réfléchie, où k est le nombre d'onde, et où $Z_c = \rho c$ est l'impédance caractéristique du milieu de propagation avec ρ la masse volumique et c la célérité des ondes. En $x = L$, la condition limite d'impédance peut s'écrire :

$$\hat{Z}(x = L) = \frac{\hat{p}(x = L)}{\hat{v}(x = L)}, \tag{2}$$

ce qui conduit à l'équation :

$$\hat{A}e^{-jkL} + \hat{B}e^{jkl} = \frac{\hat{Z}(x = L)}{Z_c}\left(\hat{A}e^{-jkL} - \hat{B}e^{jkL}\right) \tag{3}$$

En posant $\eta = \hat{Z}(x = L)/Z_c$ et $\hat{R}_p = \hat{B}/\hat{A}$, il vient :

$$\hat{A}\left((1 - \eta)e^{-jkL} + R_p(1 + \eta)e^{jkL}\right) = 0. \tag{4}$$

Cette équation possède deux solutions :

$$\begin{cases} \hat{A} &= 0 \\ \hat{R}_p &= \dfrac{\eta - 1}{\eta + 1} \quad e^{-2jkL} \end{cases} \tag{5}$$

La première solution est une solution triviale (pas de champ acoustique), et la seconde solution permet d'obtenir une expression du coefficient de réflexion.

En $x = 0$, la condition limite sur la vitesse acoustique peut s'écrire :

$$\hat{v}(x=0) = \frac{1}{Z_c}(\hat{A} - \hat{B}) = V_0. \tag{6}$$

ou encore :

$$\frac{1}{Z_c}\hat{A}(1 + \hat{R}_p) = V_0. \tag{7}$$

L'amplitude complexe $\hat{A}$ s'exprime alors uniquement en fonction de V_0 et $\hat{R}_p$ comme suit :

$$\hat{A} = \frac{V_0 Z_c}{1 - \hat{R}_p}. \tag{8}$$

L'insertion des équations 5 et 8 dans le système d'équations 9 permettent dès lors d'apporter une description détaillé du champ acoustique :

$$\begin{cases} \hat{p}(x) &= \hat{A}\left(e^{-jkx} + \hat{R}_p e^{jkx}\right) \\ \hat{v}(x) &= \dfrac{\hat{A}}{Z_c}\left(e^{-jkx} - \hat{R}_p e^{jkx}\right) \end{cases} \tag{9}$$

2.1.2 Prise en compte de la dissipation et de la dispersion

Afin de prendre en compte les phénomènes de dissipation (atténuation) et dispersion (célérité variant en fonction de la fréquence) dans le tube de Kundt, le nombre d'onde acoustique peut être réécrit comme suit :

$$k_d = \frac{\omega}{c} + (1 - i)\alpha, \tag{10}$$

où α est le coefficient d'absorption défini tel que :

$$\alpha = 3 \cdot 10^{-5}\sqrt{f}/r. \tag{11}$$

2.2 Travail à réaliser

- Écrire un code permettant de calculer, de tracer et d'animer la distribution de la pression et de la vitesse acoustique dans le tube de Kundt (Cf. exemple de la vidéo ***distribution.mp4***)
- Analyser l'évolution du coefficient de réflexion $\hat{R}_p$ en fonction du rapport η et de la fréquence f. Vous tracerez en particulier vos résultats dans un plan (η, f) afin d'avoir une vue globale de l'évolution de $\hat{R}_p$. Vous observerez les module et phase de ce coefficient, ainsi que ses parties réelle et imaginaire.
- Améliorer ce code afin de prendre en compte les pertes. Vous comparerez les résultats obtenus avec et sans pertes et mettrez en évidence, à l'aide de configurations que vous aurez préalablement déterminées, les phénomènes de dissipation et de dispersion.

Projet 3 : L'interface du reste de ma vie (étudiante !)

Travail à réaliser

En utilisant ce que vous avez vu au(x) semestre(s) précédent(s), vous allez construire une interface vous permettant de charger un fichier enregistrer avec le logiciel CTTM et d'en afficher les résultats.

- Créer des fonctions pour charger les données d'un fichier « CTTM », pour afficher des données temporelles, pour afficher des données fréquentielles.
- Développer un programme permettant à l'utilisateur d'ouvrir un fichier « CTTM » qu'il choisit et qui trace automatiquement les données.

⚠ **Les consignes ci-dessous nécessitent l'auto-apprentissage d'un nouveau module :** ⚠

- Faites en sorte qu'au lancement du programme, une interface se lance, avec des axes de tracés vide au début, où un bouton permet à l'utilisateur de charger un jeu de données.
- Ajouter à l'interface précédente la possibilité pour l'utilisateur de charger un autre jeu de données sur la même courbe.
- Créer des boutons permettant : de sauvegarder la figure, de la réinitialiser, de supprimer le dernier jeu de données rajouté, ou bien de le remettre.
- Ajouter la gestion des erreurs.
- ***optionnel :*** Ajouter à cette interface toutes fonctionnalités qui vous sembleraient utiles.

Projet 4 : Traitement du signal et imagerie

Dans ce projet, il vous est proposé de visualiser et de traiter un signal d'acoustique picoseconde mesuré par laser dans un polycristral de glace d'eau soumis à une très haute pression statique (84 GPa, ce qui correspond à la pression régnant à 2000 km sous terre). Dans ce type de mesure, une onde élastique plane de type longitudinale est générée par une première impulsion laser et est suivi au cours de sa propagation dans la glace à l'aide d'une seconde impulsion laser. Le signal obtenu est constitué d'un pic dit de coïncidence au temps $t = 0$ correspondant à la génération de l'onde élastique, d'une composante basse fréquence correspondant à la diffusion thermique de l'échauffement causée par la première impulsion laser et enfin d'une oscillation, dite oscillation de Brillouin, dont la fréquence est associée à la vitesse de propagation de l'onde élastique dans le milieu. Si le milieu est homogène, cette fréquence est constante tout au long de la propagation et donc du temps dans le signal. Si la vitesse change, cela se traduira dans le signal par un changement de la fréquence des oscillations au cours du temps.

Pour extraire la(les) fréquence(s) contenue(s) dans un signal oscillant, il convient d'en calculer le spectre fréquentiel.

Outil de traitement de signal nécessaire au projet : la transformée de Fourier

Pour calculer le spectre fréquentiel ici, et sans plus d'explications, il vous sera demandé de réaliser une transformée de Fourier numérique à l'aide de la fonction `np.fft.fft` du module `numpy`. Afin de pouvoir visualiser le spectre, il vous faudra générer l'axe des fréquences à l'aide de la fonction `np.fft.fftfreq` du même module. L'exemple qui suit illustre l'utilisation de ces deux nouvelles fonctions :

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
```

```
4  dt = 0.1     # PAS temporel (s)
5  t = dt*np.arange(256)    # Vecteur temps
6  sp = np.fft.fft(np.sin(2*np.pi*t))  # Calcul du spectre de
7                                      # la fonction sin (f=1 Hz)
8
9  NFFT = np.size(t)    # Nombre de points dans le spectre
10 freq = np.fft.fftfreq(NFFT,d=dt)     # Calcul de l'axe fréquentiel
11
12 plt.figure()         # Crée une figure
13 plt.plot(freq, np.abs(sp))  # Trace le module du spectre en
14                             # fonction de la fréquence
15 plt.title('Spectre en fréquence')    # Ajoute un titre
16 plt.xlabel('Fréquence (Hz)')     # Ajoute un nom aux abscisses
17 plt.ylabel('Module du spectre (-)') # Ajoute un nom aux ordonnées
18 plt.show()       # Affiche la figure
```

Extrait de code 1 – Exemple d'utilisation de `np.fft.fft` et de `np.fft.fftfreq`

Si vous exécutez les lignes de code précédentes, vous verrez que les fréquences vont de -5 Hz à 5 Hz et que le tracé du module du spectre en fonction de la fréquence contient un trait horizontal car les fréquences ne sont pas rangées par ordre croissant à partir de -5 Hz. On ne va pas s'attarder sur cela ici. Le spectre étant symétrique, on ne le tracera qu'entre 0 et la valeur maximale. Pour ce faire, il faudra tracer la moitié du spectre en substituant la ligne 13 de l'exemple précédent par :

```
1 # Trace le module du spectre pour les fréquences positives
2 plt.plot(freq[0:NFFT//2-1], np.abs(sp[0:NFFT//2-1]))
```

Extrait de code 2 – Ligne pour tracer le spectre uniquement sur les fréquences positives

Le but des étapes qui suivent est de vous enseigner, par l'application, la démarche de réflexion et de traitement qu'un acousticien se doit de suivre pour extraire les informations souhaitées d'un signal, que ce dernier soit audible ou ultrasonore (ici plusieurs dizaines de GHz !).

4.1 Étape 1 → visualisation de la mesure

Une fois la mesure effectuée, la première étape essentielle est de visualiser cette mesure.

Le signal à traiter est enregistré dans le fichier `time_rawsignals_84GPa_nice.txt` déposé sur UMTICE. Ce fichier est constitué d'un en-tête suivi de deux colonnes, la première correspondant au temps et la seconde à l'amplitude.

Travail à réaliser

1. Écrire une fonction que vous nommerez `load_ExpData` et qui prendra deux arguments d'entrée non optionnels : le nom du fichier à charger et le chemin absolu vers son lieu de stockage sur le disque dur. Cette fonction devra retourner deux vecteurs de type `ndarray` : `time` qui comprendra le vecteur temps (en ns) et `amp` qui comprendra le vecteur amplitude.
2. Écrire un programme qui demande à l'utilisateur de sélectionner le fichier texte contenant les données à traiter, qui charge les données contenus dans le fichier texte avec la fonction `load_ExpData` et qui trace le signal en utilisant l'API *orientée objet*.

4.2 Étape 2 → extraction des informations sur la mesure brute

La deuxième étape, avant même de mettre en œuvres des méthodes de traitement du signal avancée, est d'essayer d'extraire les informations recherchées directement sur les données brutes mesurées. Cette étape est dans bien des cas suffisantes.

Travail à réaliser / Questions

1. Compléter le programme précédent afin de calculer le spectre en fréquence du signal et l'axe des fréquences.
2. Modifier la partie du programme affichant les résultats afin d'afficher sur deux sous-figures d'une même figure le signal temporel d'une part et le module du spectre fréquentiel en dB d'autre part. (La valeur de référence pour le calcul des dB sera la valeur maximale du module du spectre.)
3. Sachant que l'unité de l'axe temporel est la nanoseconde, quelle est l'unité de l'axe fréquentiel ? Veiller à bien renseigner ces unités sur vos axes.
4. Quel est le contenu fréquentiel des oscillations présentes dans le signal ? Que cela signifie-t-il d'après l'introduction du projet ?
5. Au lieu de réaliser la transformée de Fourier sur l'ensemble du signal, il est maintenant proposer d'en sélectionner une partie et de calculer le spectre fréquentiel de cette partie uniquement. Calculer les spectres du signal pour les trois intervalles temporels suivants : $t \in [0,15; 0,4]$ ns, $t \in [0,45; 0,7]$ ns et $t \in [0,9; 2]$ ns. Commenter.
6. Tester à nouveau votre programme en sélectionnant le fichier déposé sur UMTICE nommé `time_rawsignals_84GPa_bad.txt`.
7. Vérifier que les oscillations sont toujours bien présentes dans le second signal. Pourquoi, selon vous, sont-elles plus difficiles à observer sur ce signal temporel par rapport au premier ?

On remarque, en comparant les résultats obtenus avec les deux signaux que lorsque l'amplitude des oscillations est faible par rapport à celle du pic de coïcidence sur le signal temporel, alors, dans le spectre fréquentiel, l'amplitude des pics associés aux fréquences des oscillation est également faibles. Afin de faciliter l'extraction automatique des fréquences des oscillations, il est généralement entrepris d'appliquer un pré-traitement au signal avant de réaliser la transformée de Fourier.

4.3 Étape 3 → extraction des informations sur la mesure pré-traitée

La troisième étape, dans le cas où l'extraction des informations sur la mesure brute se révèle peu pratique, est de pré-traiter le signal. Dans le cas des signaux qui nous intéressent, le but d'un tel traitement revient à chercher à retirer du signal mesuré la basse fréquence. Pour ce faire, on peut appliquer un filtrage fréquentiel ou chercher un signal basse fréquence à soustraire au signal mesuré.

Il est ici proposé de créer une fonction `pre_treatment` qui prendra quatre arguments d'entrée, dont deux non optionnels, les vecteurs `time` et `amp`, et deux optionnels `meth`, une chaîne de caractère permettant de choisir la méthode de pré-taitement à utiliser et qui prendra comme valeur par défaut `'pol_fit'`, et une liste de paramètres `lst_param` nécessaires à la méthode utilisée qui vaudra par défaut `[0, 20]` (où 0 correspond au temps à partir duquel commencé le fit par un polynôme et 20 correspond au degré du polynôme utilisé pour le fit).

Tous les développements seront testés pas à pas puis intégrés dans la fonction `pre_treatment`. Chaque point ci-dessous correspondra à une méthode de pré-traitement à développer. Nous vous conseillons d'utiliser le signal `time_rawsignals_84GPa_nice.txt` pour vos tests.

Travail à réaliser

1. Pour filtrer les basses fréquences d'un signal, une méthode simple et rapide est de calculer la dérivée de ce signal. Expliquer pourquoi cela filtre les basses fréquences selon vous et réaliser cette dérivation numériquement de deux façons : (i) en le codant vous-même, puis (ii) en utilisant la fonction `np.diff`. La chaîne de caractère associée à cette méthode sera `meth='diff'`. *indice : une dérivée est, à quelque chose près, la différence entre deux points successifs...*
2. Pour filtrer un signal, on peut également utiliser un filtrage passe bande, par exemple pour ne garder que les fréquences du signal comprises entre 40 GHz et 150 GHz. Pour ce faire, vous aurez besoin d'importer le sous module `signal` de `scipy` pour utiliser les fonctions `sig.butter` pour définir les paramètres du filtre et `sig.filtfilt` pour appliquer le filtre sur le signal. La chaîne de caractère associée à cette méthode sera `meth='filt'`.
3. Pour filtrer les basses fréquences d'un signal, une autre méthode peut être de soustraire en chaque point du signal une valeur moyenne calculée sur une fenêtre centrée sur le point et dont la largeur jouera sur les fréquences qui seront retirées du signal : fenêtre très large = filtrage des très basses du signal uniquement, fenêtre très fine = filtrage de toutes fréquences jusqu'aux très hautes. Il vous est demandé ici d'implémenter cette méthode basée sur une moyenne dite glissante. La moyenne étant calculée sur le point central de la fenêtre, il faudra faire attention aux premiers et derniers points du signal pour lesquels la fenêtre n'appartient plus au signal. La chaîne de caractère associée à cette méthode sera `meth='moy_glis'`.

   La forme de la basse fréquence à soustraire dans le cas spécifique des signaux étudiés ici à la forme d'une exponentielle décroissante. Les deux méthodes qui suivent proposent de trouver une fonction qui s'ajuste convenablement sur cette forme et la soustrait au signal.
4. Pour réaliser l'ajustement, on peut utiliser une fonction polynomiale en commençant l'ajustement à partir du temps pour lequel le signal est maximal. Les fonctions à utiliser pour se faire sont : `np.poly1d` et `np.polyfit`. La chaîne de caractère associée à cette méthode sera `meth='pol_fit'`.
5. *Optionnel :* Sachant que théoriquement cette décroissance est exponentielle, on peut également chercher à ajuster une fonction du type $a\exp[-b(t-t_0)]+c+d\exp[-e(t-t_0)]$, où t_0 est le temps pour lequel le signal est maximal. L'ajustement se réalise ici en utilisant la fonction `scipy.optimize. curve_fit` du sous module `optimize` du module `scipy` dont vous trouverez un exemple d'utilisation en ligne. [méthode `meth='exp_fit'`].
6. Intégrer toutes les méthodes développées dans la fonction `pre_treatment` telle que en introduction du sous-projet et intégrer l'appel à cette fonction dans le programme principal. Utiliser le signal `time_rawsignals_84GPa_nice.txt` pour tester la bonne marche de la fonction, puis avec `time_rawsignals_84GPa_bad.txt` dans un second temps et commenter (notamment en comparant avec le cas sans pré-traitement).

4.4 Étape 4 → imagerie

Tout ce qui a été mis en place peut maintenant être appliqué au cas d'une multitude de signaux collectés en différents points d'un même échantillon.

Travail à réaliser

En utilisant les signaux contenus dans le dossier « Image » sur UMTICE et ce que vous avez développé jusqu'à présent, reproduire la figure 5. Le blanc correspond à des points où aucun signal n'a été mesuré : ces signaux ont pour valeurs -1000 mV quelque soit le temps. Les mesures ont été

réalisées sur un quadrillage de 60x60 points, avec un pas spatial de 0.5 µm dans les deux directions. Certains des signaux contiennent deux fréquences comme vous le verrez. L'image est obtenue en regardant les fréquences entre 57 GHz et 73 GHz. Attention, le temps contenu dans les fichiers textes doit être redéfini (cela vient de la méthode de mesure). La bonne déclaration du vecteur temps est la suivante :

```
DeltaF = 500; % Hz
Freq_laser = 42e6; % Hz
f_sampling = 25e6; % Hz

dt = DeltaF/Freq_laser/f_sampling; % s
dt = dt*1e9; % ns

time = dt*float(np.arrange(Nt-1)); % ns
```

Extrait de code 3 – Déclaration du vecteur temps pour reproduire la figure 5

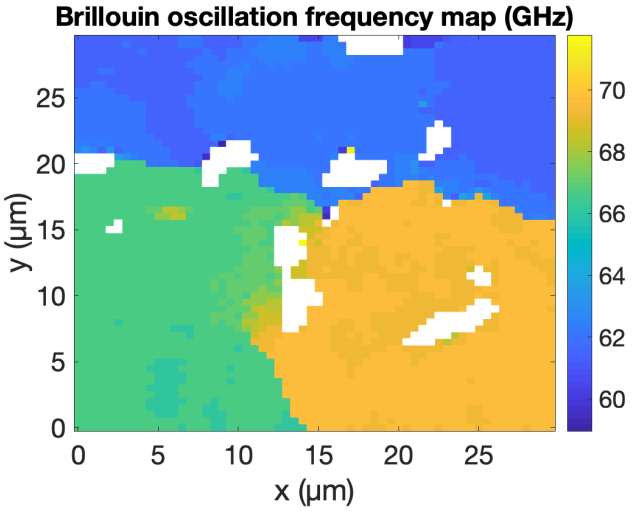

FIGURE 5 – Image de fréquence Brillouin à reproduire.

Projet 5 : Optique géométrique

Travail à réaliser

Le but de ce projet est d'illustrer les notions de stigmatisme et de stigmatisme approché pour un miroir sphérique (concave et convexe) et une lentille sphérique (biconcave et biconvexe). Il faudra montrer la différence entre un système stigmatique (approché) et un système non stigmatique. Chaque surface réflechissante sphérique sera caractérisée par son sommet, son centre (donc le rayon) et son diamètre d'ouverture. Il en va de même pour un dioptre, le paramètre supplémentaire étant l'indice de réfraction des deux milieux en contact au niveau du dioptre. Un miroir sphérique est constitué d'une seule surface réflechissante. Une lentille sphérique est constituée de deux dioptres. Pour rappel, un rayon se réflechissant sur un miroir repart avec le même angle par rapport à la normale à la surface du miroir au point d'impact. Dans le cas d'une lentille, comme il y a changement de milieu, il se produit deux réfraction (une sur chaque dioptre), la loi permettant de connaître l'angle réfracté i_2 connaissant l'angle incident i_1 et les indices de réfractions des milieux 1 (n_1) et 2 (n_2) est : $n_1 \sin i_1 = n_2 \sin i_2$.

Projet 6 : A vous de jouer maintenant [non obligatoire]

Travail à réaliser

Dans ce projet, c'est à vous de proposer le cahier des charges pour un programme, pour une interface, pour ... et à vous de proposer une solution (évidemment !).

Projet 7 : Python, ça pilote aussi... [non obligatoire]

Travail à réaliser

- Trouver comment communiquer avec une carte N.I. avec Python et créer des fonctions pour émettre un signal et pour le mesurer.
- Développer un programme permettant de faire l'acquisition d'un signal temporel.
- Développer un programme permettant de faire l'acquisition d'une réponse en fréquence.
- Développer une interface graphique permettant à un utilisateur de paramétrer une acquisition et de choisir entre une mesure temporelle ou fréquentielle.
- Intégrer à cette interface toutes fonctionnalités qui vous sembleraient utiles lors d'une acquisition avec une carte N.I.