Le Mans Université

TP de simulation numérique

Bruno Brouard

Cyril Desjouy

Samuel Raetz

L2 Sciences Pour l'Ingénieur

Année 2018 – 2019

Méthode de travail et évaluation

Les sous-projets sont annotés par difficulté suivant l'échelle suivante :

- ⋆ **Niveau 0 :** basique (non évalué à l'examen, exercices servant de remise à niveau)
- ⋆ **Niveau 1 :** simple
- ⋆⋆ **Niveau 2 :** avancé
- ⋆⋆⋆ **Niveau 3 :** difficile
- ⋆⋆⋆⋆ **Niveau 4 :** expert

Vous serez évalué en fin de semestre de **manière individuelle** via un examen qui aura lieu certainement pendant la dernière séance de TP.

Votre évaluation sera à la fois **quantitative** [1] ET **qualitative** [2].

Avant cette évaluation vous devrez remplir une feuille attestant des projets, sous-projets que vous avez réussi à terminer. Plus vous avez d'étoiles (⋆), mieux c'est ! (Mais attention il y aura un **minimum** d'étoiles à cocher sous peine d'avoir 0/20 comme note finale.)

Vous serez alors interrogé sur un ou des sous-projets [3] que vous avez indiqué comme terminés. Vous aurez pour cet examen droit à 0, 1 ou 2 jokers [4]. Plus vous avez coché d'étoiles, plus vous avez la possibilité de refuser d'être interrogé sur un sous-projet donné à l'examen.

Votre note sera un mélange entre votre note *quantité de projets terminés* (en terme d'étoiles ⋆) et vos résultats à l'examen terminal ou bien 0/20 en cas de travail insuffisant.

1. plus vous faites de projets mieux c'est !
2. un projet mal maîtrisé est très pénalisant !
3. approximativement le même nombre d'étoiles (⋆) pour tous les étudiants
4. nombre proportionnel au nombre de projets cochés comme réalisés

Pensez à montrer votre travail à l'encadrant en TP pour qu'il vous donne un maximum de conseils et valide *avec vous* vos projets (vérifie que c'est bien le travail qu'on attendait de vous).

Les projets peuvent se faire à priori dans N'IMPORTE QUEL ORDRE. Commencez par ceux qui vous motivent le plus.

Des projets ou des sous-projets pourraient être ajoutés en cours de semestre. Si vous avez des idées de projet, n'hésitez pas à nous en faire part.

Table des matières

1 Rappels de L1 **7**

1.1 Objets et Entrée/Sortie standard ⋆ 7

1.2 Les modules numpy et matplotlib ⋆ 8

1.3 Les instructions composées ⋆ 9

1.4 Fonctions et compréhensions ⋆ 9

2 Python et la bibliothèque standard **10**

2.1 Générateur aléatoire ⋆ 10

2.2 Chiffres arabes en chiffres romains ⋆⋆⋆ 11

2.3 Chiffrement sans clé de César ⋆⋆⋆ 11

2.4 Chiffrement avec clé de Vernam ⋆⋆⋆⋆ 12

3 Les figures avec `matplotlib` **14**

3.1 Les *ticks* et autres décorations ⋆ 16

3.2 Les subplots ⋆⋆ 17

3.3 Ondes stationnaires et progressives ⋆⋆ 18

4 Les animations avec `matplotlib` **20**

4.1 Une animation simple ⋆⋆ 22

4.2 Une animation avancée ⋆⋆⋆ 22

5 Les widgets avec `matplotlib` **23**

5.1 Un widget simple ⋆⋆ 23
5.2 Un widget avancé ⋆ ⋆ ⋆ 25

6 Lecture de fichiers **26**
6.1 Test de répétabilité : statistiques simples ⋆⋆ 26
6.2 Analyse de données expérimentales : tube de Kundt ⋆ ⋆ ⋆ 28
6.3 Lecture de fichiers audio sous Python ⋆ ⋆ ⋆ 31

7 Diagramme de directivité **34**
7.1 Tracé de diagramme de directivité ⋆⋆ 34

Projet 1

Rappels de L1

Introduction

Le but de ce projet est de tester vos connaissances sur les concepts de base que vous avez vus en L1 SPI [1]. Les notions abordées dans ce projet sont les notions essentielles sans lesquelles vous ne pourrez mener à bien les projets proposés dans ces TP. Les trois sous-projets proposés ci-dessous doivent vous paraître extrêmement simples. Si ce n'est pas le cas, il vous faudra retravailler les cours/TD de L1 chez vous.

Sous-projet : 1.1 Objets et Entrée/Sortie standard ⋆

Écrire un programme qui :

- Demande à l'utilisateur de saisir un texte (chaîne de caractères) que vous assignerez à la variable `s1` qui référencera un objet du type `str`.
- Demande à l'utilisateur de saisir un nombre réel (nombre à virgule) que vous assignerez à la variable `f` qui référencera un objet du type `float`.
- Crée une nouvelle chaîne de caractères que vous nommerez `s2` ayant la même valeur que `s1` mais dans laquelle toutes les voyelles sont remplacées par le caractère `?`. Vous utiliserez pour ce faire la méthode `replace` hérité par les objets de type `str`.
- Affiche la partie entière de `f`.
- Multiplie `s2` par la partie entière de `f`
- Affiche le résumé des opérations effectuées sur la sortie standard.

Votre programme s'exécutera comme suit :

1. Pour les étudiants n'ayant pas suivi la L1 SPI au Mans et ne connaissant pas le langage Python, il est impératif de vous mettre à jour en dehors de ces séances de TP en lisant les supports de cours et en travaillant les notebooks disponibles sur UMTICE.

```
Entrez un texte : Ceci est un test
Entrez un nombre réel : 2.3
La partie entière de 2.3 est 2.
La chaine "Ceci est un test" sans voyelles répétée 2 fois prend la forme :
C ?c ? ?st ?n t ?stC ?c ? ?st ?n t ?st
```

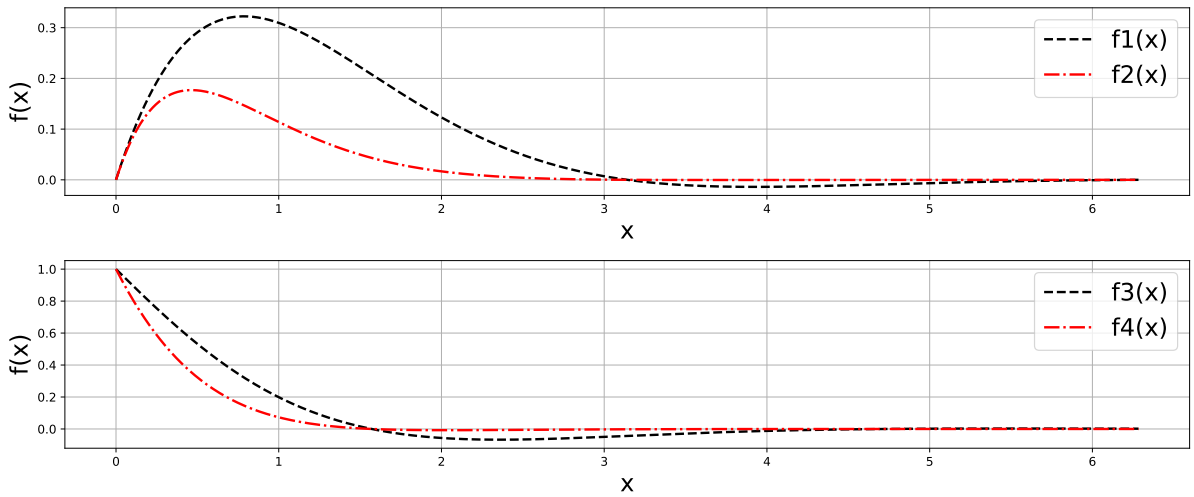
Sous-projet : 1.2 Les modules numpy et matplotlib ⋆

On considère les fonctions $f_1(x)$ à $f_4(x)$ définies telles que :

$$\begin{cases} f_1(x) &= \sin(x) \cdot e^{-x}, \\ f_2(x) &= \sin(x) \cdot e^{-2x}, \\ f_3(x) &= \cos(x) \cdot e^{-x}, \\ f_4(x) &= \cos(x) \cdot e^{-2x}, \end{cases} \tag{1.1}$$

sur l'intervalle $x \in [0, 2\pi]$. Écrire un programme qui :

- Importe les modules nécessaires.
- Crée l'axe des abscisses. Celui-ci contiendra 1000 valeurs équiréparties entre 0 et 2π.
- Calcule les fonctions $f_1(x)$ à $f_4(x)$.
- Créer une figure divisée en deux sous-figures présentant les 4 fonctions comme présenté à la figure 1.2
- Sauvegarde la figure en pdf.



Sous-projet : 1.3 Les instructions composées ⋆

Écrire un programme qui reproduit et affiche la matrice présentée ci-dessous sur la sortie standard. Votre programme utilisera des structures itératives et conditionnelles.

```
[[99  8  8  8  8  8  8  8  8  8]
 [ 1 99  8  8  8  8  8  8  8  8]
 [ 1  1 99  8  8  8  8  8  8  8]
 [ 1  1  1 99  8  8  8  8  8  8]
 [ 1  1  1  1 99  8  8  8  8  8]
 [ 1  1  1  1  1 99  8  8  8  8]
 [ 1  1  1  1  1  1 99  8  8  8]
 [ 1  1  1  1  1  1  1 99  8  8]
 [ 1  1  1  1  1  1  1  1 99  8]
 [ 1  1  1  1  1  1  1  1  1 99]]
```

Sous-projet : 1.4 Fonctions et compréhensions ⋆

Écrire une fonction que vous nommerez `pair` et qui prendra deux arguments d'entrée optionnels `mini` et `maxi` ayant pour valeurs par défaut respectives 0 et 10 et qui retournera une liste des entiers pairs entre `mini` et `maxi`. Votre fonction fera au maximum deux lignes et s'exécutera comme suit :

```
pair()
[0, 2, 4, 6, 8, 10]

pair(4, 14)
[4, 6, 8, 10, 12, 14]
```

Projet 2

Python et la bibliothèque standard

Introduction

Ce projet consiste à mettre en application vos connaissances du langage Python à travers des exercices n'utilisant que la base du langage et quelques modules standards. Vous utiliserez en particulier les modules `random`, `string` et `os` et renforcerez vos connaissances sur les fonctions, les classes, les modules, la manipulation de chaînes de caractères et les notions d'algorithmique classiques.

Sous-projet : 2.1 Générateur aléatoire ⋆

Il s'agit ici d'utiliser la méthode `choice` du module `random` de la bibliothèque standard afin de créer une fonction que vous nommerez `key` permettant de générer une chaîne de caractères composée de N lettres aléatoires choisies parmi les 26 lettres de l'alphabet latin. Cette fonction prendra en argument d'entrée un entier représentant la longueur de la chaîne de caractères à générer. Vous utiliserez également le module `string` de la bibliothèque standard qui fournit notamment la constante `ascii_lowercase` qui vous sera utile pour cet exercice. Votre fonction `key` ne fera pas plus de 2 lignes et s'exécutera comme illustré sur l'exemple suivant :

```
key(10)
fhadokphud
```

Sous-projet : 2.2 Chiffres arabes en chiffres romains ⋆⋆⋆

Les nombres en chiffres romains s'écrivent avec les lettres de l'alphabet latin. La table 2.1 suivante regroupe les correspondances entre les chiffres romains et les chiffres arabes.

arabe	I	V	X	L	C	D	M
romain	1	5	10	50	100	500	1000

TABLE 2.1 – Correspondance entre chiffres arabes et romains.

Travail à réaliser

- Écrire une fonction `arabic2roman` permettant de convertir un nombre arabe en nombre romain. Votre fonction sera capable de convertir les nombres entre 1 et 4999.
- Écrire une fonction `roman2arabic` permettant de convertir un nombre romain en nombre arabe.

Sous-projet : 2.3 Chiffrement sans clé de César ⋆⋆⋆

Le chiffrement, qu'on appelle plus généralement cryptage, est un procédé permettant de rendre la compréhension d'un texte impossible à toute personne ne connaissant pas la méthode de décryptage. Il existe de nombreux procédés de cryptage. Dans le cadre de ce sous-projet nous allons étudier le *chiffre de César*, et plus précisément un cas particulier de cette méthode nommé *ROT13* qui consiste à décaler de 13 lettres chaque lettre d'un document à chiffrer. La table 2.2 présente la correspondance entre les caractères cryptés et les caractères non cryptés pour la méthode *ROT13*.

clair	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
crypté	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m

TABLE 2.2 – Correspondance entre caractères cryptés et non cryptés pour le *ROT13*.

C'est ce type de code qu'utilisait Jules César dans ses correspondances secrètes [1]. Le défaut de ce cryptage est qu'il ne s'occupe que des lettres. Les symboles, la ponctuation et les caractères accentués ne sont pas cryptés. À l'époque romaine, les chiffres romains utilisaient l'alphabet latin et pouvaient être cryptés facilement. Il convient aujourd'hui, pour crypter un message

1. Jules César utilisait un décalage de 3 lettres dans ses correspondances secrètes.

contenant du texte et des chiffres, de convertir les chiffres arabes en chiffres romains ou de les écrire en toutes lettres. Notons que ce type de cryptage est qualifié de cryptage *faible* car toute personne connaissant ce procédé de cryptage peut décrypter très facilement n'importe quel message.

Travail à réaliser

Il s'agit ici d'écrire une fonction que vous nommerez `ROT13` prenant en argument d'entrée obligatoire une chaîne de caractères (pouvant être indifféremment constituée de caractères minuscules ou majuscules) et en arguments d'entrée optionnels :

- la clé `n` ayant pour valeur par défaut 13, mais pouvant prendre toutes les valeurs entre 1 et 26. Si la valeur de `n` n'est pas comprise dans cet intervalle, la fonction affichera un message d'erreur.
- la clé `way` ayant pour valeur par défaut *encrypt* qui permet d'utiliser la fonction pour crypter, et qui peut également prendre la valeur *decrypt* permettant d'utiliser la fonction pour décrypter.

Votre fonction `ROT13` s'exécutera comme l'illustre l'exemple suivant :

```
ROT13('Ceci est une phrase secrète')
Prpv rfg har cuenfr frpergr

ROT13('Prpv rfg har cuenfr frpergr', way='decrypt')
Ceci est une phrase secrete

ROT13('Ceci est une phrase secrète', n=28)
Chiffrement en rotation sur 28 caractères impossible.
La valeur de n doit être inférieure ou égale à 26.
```

Sous-projet : 2.4 Chiffrement avec clé de Vernam ⋆⋆⋆⋆

Les méthodes de cryptage modernes s'appuient généralement sur l'utilisation de clés de chiffrement. Il existe deux catégories de systèmes de chiffrement :

- le chiffrement symétrique qui utilise la même clé pour chiffrer et déchiffrer,
- le chiffrement asymétrique qui utilise une paire de clés appelées *clé publique*, servant au chiffrement, et *clé privée*, servant à déchiffrer.

La méthode étudiée dans le cadre de ce sous-projet est la méthode du *masque jetable* également appelée *chiffre de Vernam*. Cette méthode est une méthode de chiffrement symétrique qui utilise une clé dont la longueur doit être strictement égale à la longueur du message à chiffrer[2]. La

2. Le chiffre de Vernam impose également que les caractères constituant la clé soit choisis de façon aléatoire et que la clé ne soit utilisée qu'une unique fois, d'ou l'appellation *masque jetable*.

méthode de chiffrement est la suivante : un nombre est attribué à chaque lettre de l'alphabet (de 1 à 26). La valeur de chaque lettre du message à crypter est ensuite ajoutée à la valeur correspondante dans la clé. Si le résultat est strictement supérieur à 26, on soustrait alors 26. La méthode de déchiffrement est similaire à la méthode de chiffrement à la différence qu'on retranche ici la valeur de la clé au lieu de l'ajouter. Si la valeur obtenue est négative ou nulle, il faut alors ajouter 26.

Travail à réaliser

1. Écrire une classe que vous nommerez `Vernam` qui :
   - prend en argument d'entrée la clé de cryptage (de type `str`),
   - fournit une méthode `encrypt` prenant en argument d'entrée le message à crypter (de type `str`) et retournant le message crypté. Si la clé de cryptage et le message à crypter ont des tailles différentes, cette fonction affichera un message d'erreur,
   - fournit une méthode `decrypt` prenant en argument d'entrée le message à décrypter (de type `str`) et retournant le message décrypté. Si la clé de cryptage et le message à décrypter ont des tailles différentes, cette fonction affichera également un message d'erreur.
2. Améliorer cette classe `Vernam` en permettant la génération automatique d'une clé (lors du cryptage uniquement) à l'aide de la fonction `key` développé précédemment. La méthode `encrypt` pourra par exemple prendre un argument optionnel nommé `keygen` de type booléen précisant si une clé doit être générée ou non.
3. Assembler tous les développements menés à bien dans ce projet dans un module que nous nommerez `crypto` qui fournira :
   - La fonction `key`,
   - Les fonctions `arabic2roman` et `roman2arabic`,
   - La fonction `ROT13`
   - La classe `Vernam`

Indices :

- *Vous pourrez utiliser des compréhensions de dictionnaire pour avoir des objets vous permettant de faire les correspondances {chiffre : lettre} et {lettre :chiffre}.*
- *Vous pourrez utiliser la fonction `zip` afin de créer un itérable contenant (clé, message) afin d'itérer simultanément sur ces deux valeurs au sein d'une même boucle.*

Projet 3

Les figures avec `matplotlib`

Introduction

En sciences, il est courant de présenter des résultats sous forme graphique. Sous Python, un package très utilisé pour tracer des figures est le package `matplotlib`.

Ce package propose plusieurs API (*Application Programming Interface*) dont l'API `pyplot` et l'API *orientée objet*. L'API pyplot est celle que vous avez largement utilisée en première année. Un exemple basique de l'utilisation de cette API est illustré par l'extrait de code 3.1. Cette API propose l'utilisation directe des fonctions fournies par le sous module `pyplot` de `matplotlib`. Une fois un objet de type `figure` crée (grâce à la fonction `pyplot.figure()`), l'utilisation des fonctions fournies par `pyplot` permet de réaliser des actions basiques sur la figure (ajout de titre, de noms aux axes, ...). Se voulant simple d'utilisation, cette API est généralement utilisée pour créer des figures basiques de manière interactive ou programmée. Lorsqu'il s'agit de créer des figures plus élaborées, il faudra utiliser l'API *orientée objet* qui fournit un ensemble d'outils de contrôle et de personnalisation.

```
1 import matplotlib.pyplot as plt
2
3 plt.figure()          # Crée une figure
4 plt.plot(x, y)        # Trace y en fonction de x
5 plt.title('Titre')    # Ajoute un titre
6 plt.xlabel('x')       # Ajoute un nom aux abscisses
7 plt.ylabel('y')       # Ajoute un nom aux ordonnées
8 plt.show()            # Affiche la figure
```

Extrait de code 3.1 – API pyplot

Python est un langage orienté objet, et il va de soi que la bibliothèque `matplotlib` l'est

également. L'extrait de code 3.2 présente le principe d'utilisation de l'API *orientée objet*. Il s'agit pour cette API d'utiliser les méthodes et attributs hérités par les objets de types `axes` et `figure` de `matplotlib`. Pour mettre en forme une nouvelle figure en utilisant l'API *orientée objet*, il faut :

1. Créer un objet `figure` et un ou plusieurs objets `axes` grâce à la fonction `pyplot.subplots` (à ne pas confondre avec la fonction `pyplot.subplot` utilisée pour créer des sous-figures avec l'API `pyplot`).
2. Travailler directement sur les objets `figure` et `axes` en utilisant leurs méthodes.
3. Afficher votre (ou vos) figure(s) à l'aide de la fonction `pyplot.show()`.

```
1 import matplotlib.pyplot as plt
2
3 fig, ax = plt.subplots() # Crée des objets figure et axes
4 ax.plot(x, y)            # Utilise la méthode plot héritée par axes
5 ax.set_title('Titre')    # Utilise la méthode set_title héritée par axes
6 ax.set_xlabel('x')       # Utilise la méthode set_xlabel héritée par axes
7 ax.set_ylabel('y')       # Utilise la méthode set_ylabel héritée par axes
8 plt.show()               # Affiche la figure
```

Extrait de code 3.2 – API objet

Il est possible de préciser le nombre d'objets `axes` (i.e. de sous figures) lors de la création d'une figure. L'extrait de code 3.3 illustre le principe de création de sous figures avec l'API *orientée objet*. La fonction `pyplot.subplots` est dans cet exemple appelée avec les arguments (2, 2) qui signifient que la figure sera composée de 4 sous figures (2x2) disposées sur 2 lignes et 2 colonnes. L'objet `ax` est alors un objet de type `ndarray` contenant les 2x2, soit 4 objets de type `axes` associées aux 4 sous-figures imposées lors de l'initialisation de notre figure.

```
1 import matplotlib.pyplot as plt
2
3 fig, ax = plt.subplots(2,2)      # Crée 1 figure avec 4 axes (2x2)
4 ax[0, 0].plot(x, y1)             # line 0, col 0 : 1ère sous-figure
5 ax[0, 1].plot(x, y2)             # line 0, col 1 : 2ème sous-figure
6 ax[1, 0].plot(x, y3)             # line 1, col 0 : 3ème sous-figure
7 ax[1, 1].plot(x, y4)             # line 1, col 1 : 4ème sous-figure
8 plt.show()                       # Affiche la figure
```

Extrait de code 3.3 – API objet

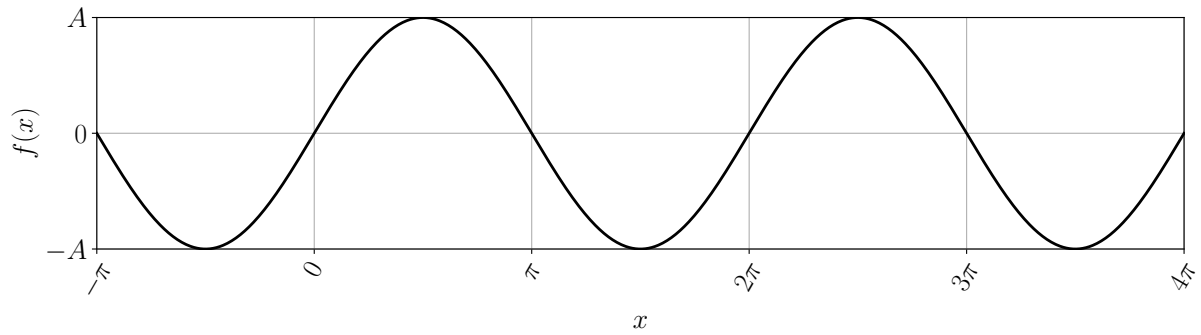
Les objets `axes` contiennent la majorité des éléments de la figure (axes, labels, tracés, textes...) tandis que les objets `figure` contiennent plutôt des méthodes générales s'appliquant à l'ensemble de la figure (ajout d'objets `axes`, apparence générale, taille...). Les objets

de types `axes` et `figure` héritent de très nombreuses et puissantes méthodes que vous pourrez retrouver sur leurs pages de manuel associées : matplotlib.axes et matplotlib.figure

Sous-projet : 3.1 Les *ticks* et autres décorations ⋆

On considère ici la fonction $f(x) = sin(x)$ sur l'intervalle $[-\pi, 4\pi]$. Le but de cet exercice est de reproduire la figure 3.1 à l'aide de l'API `pyplot` et de l'API *orientée objet*.

Deux particularités graphiques sont à apprécier sur cette figure 3.1. La première est le texte formaté en LaTeX [1]. La seconde est la personnalisation des axes des abscisses et ordonnées, et en particulier des labels qui les constituent.



Écrire du LaTeXsous `matplotlib`

La bibliothèque `matplotlib` permet l'utilisation de LaTeX pour formater du texte sur une figure. Pour ce faire, il suffit de préfixer les chaînes de caractères à formater avec le caractère "r" (pour raw string). Tout code LaTeX inséré dans un raw string sera alors interprété par LaTeX (si l'option est activé par défaut sur votre système). À titre d'illustration, l'extrait de code suivant :

```
1     plt.xlabel(r'Une fonction $f(\theta)$')
```

permet de formater le label des abscisses comme suit :

Une fonction $f(\theta)$

1. LaTeX est un langage de programmation permettant de créer et de mettre en forme des documents texte. Ce langage est extrêmement populaire et très largement utilisé dans la communauté scientifique pour l'écriture d'articles, de thèses, de rapports ou la création de présentations scientifiques.

Personnaliser les axes

Par défaut, lorsqu'une figure est crée, les axes des abscisses et des ordonnées ainsi que les valeurs affichées sur ces derniers sont ajustés automatiquement. Il arrive souvent que le résultat final ne corresponde pas au résultat souhaité. La bibliothèque `matplotlib` permet de personnaliser facilement ces axes grâce :

- aux fonctions `pyplot.xticks` et `pyplot.yticks` comme suit :

```
plt.xticks([liste des positions], [liste des labels (str)], options)
```

- ou aux méthodes `set_xticks` et `set_yticks` associées à `set_xticklabels` et `set_xticklabels` héritées par les objets `axes` comme suit :

```
ax.set_xticks([liste des positions])
ax.set_xticklabels([liste des labels (str)], options)
```

L'une des `options` intéressante est l'option `rotation` qui permet d'orienter le texte suivant un angle précisé. Toutes les options sont présentées sur la page de manuel dédié aux objets `.Text` : Propriété du texte sous `matplotlib`.

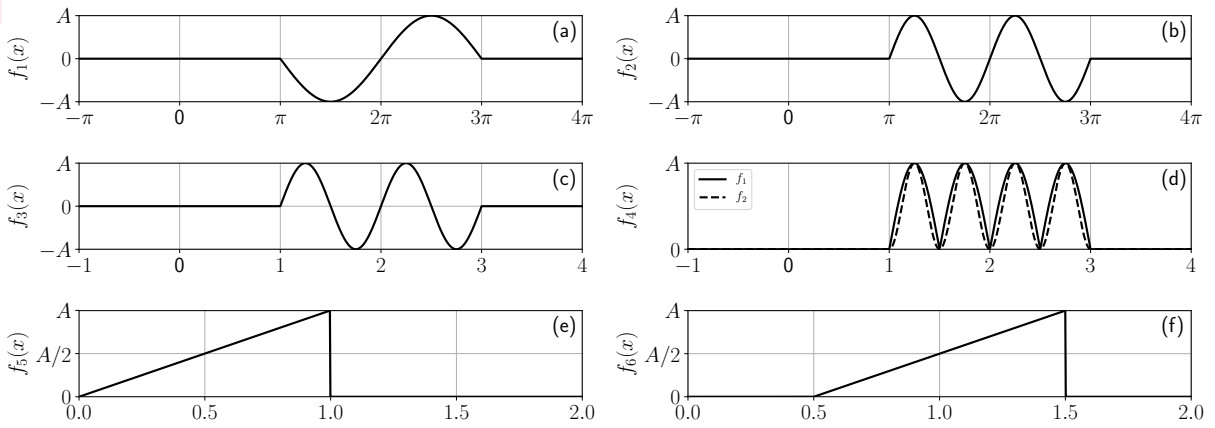
Travail à réaliser

1. Reproduire la figure 3.1 à l'aide de l'API `pyplot`
2. Reproduire la figure 3.1 à l'aide de l'API *orientée objet*

Sous-projet : 3.2 Les subplots ⋆⋆

En TD d'acoustique (TD1), vous avez établi la formulation mathématique de chacun des six signaux présentés sur la figure 3.2. Il s'agit ici de reproduire cette figure 3.2 en utilisant l'API *orientée objet*. Vous optimiserez la longueur de votre programme notamment en utilisant des boucles `for` permettant d'ajuster les propriétés communes à toutes les sous-figures [2].

2. A titre d'exemple, la grille est affichée sur chacune des sous-figures. Il semble donc pertinent d'appliquer cette transformation à toutes les sous-figures en utilisant une boucle plutôt que d'appeler la fonction/méthode `grid()` pour chaque sous-figure !

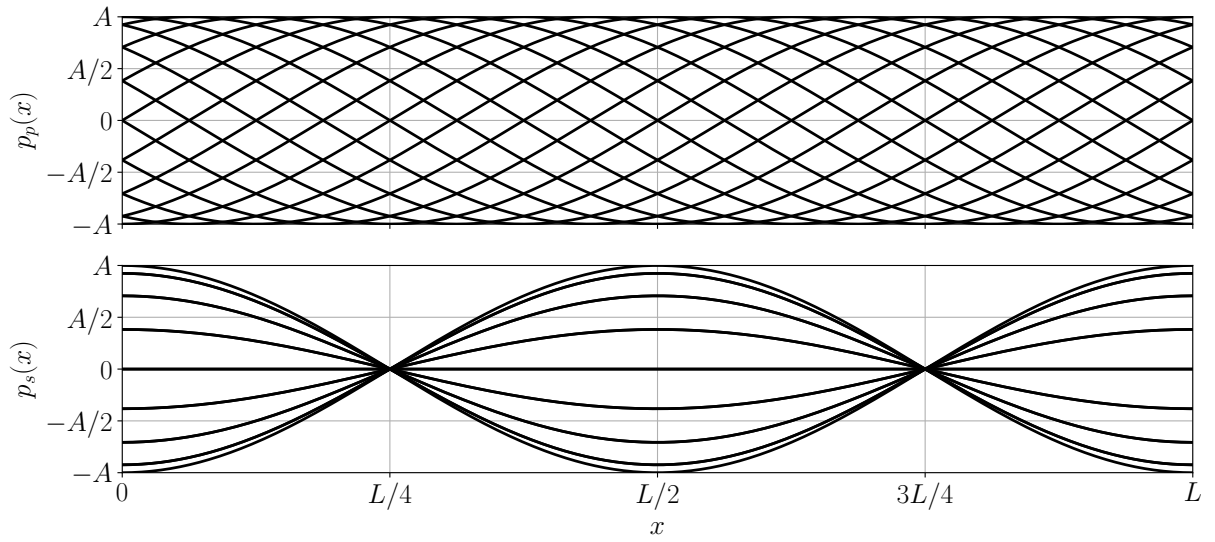


Sous-projet : 3.3 Ondes stationnaires et progressives ⋆⋆

En acoustique, selon les conditions imposées aux limites du domaine de propagation, plusieurs types d'ondes peuvent se propager. Deux cas singuliers ont été étudiés en TD d'acoustique cette année. Il s'agit de l'onde stationnaire et de l'onde progressive. Les formes générales de la pression acoustique en 1 dimension correspondant à ces deux types d'onde sont les suivantes :

$$\begin{cases} \tilde{p}_s(x,t) = & A\cos(kx)\cos(\omega t) & \text{pour une onde stationnaire pure.} \\ \tilde{p}_p(x,t) = & A\cos(\omega t - kx) & \text{pour une onde progressive pure.} \end{cases} \tag{3.1}$$

La figure 3.3 représente ces deux types d'onde à plusieurs instants t pour des abscisses compris entre 0 et L =1 m, une fréquence f = 340 Hz et une célérité c = 340 m/s.



Travail à réaliser

Il s'agit ici de reproduire la figure 3.3. Cette figure présente les formes de l'onde de pression pour 16 instants répartis linéairement sur une période temporelle de l'onde acoustique. Vous utilisez des boucles `for` afin de tracer les formes d'onde à ces 16 instants et veillerez à correctement libeller les axes de la figure.

Projet 4

Les animations avec `matplotlib`

Introduction

La bibliothèque `matplotlib` fournit deux fonctions principales pour créer des animations, à savoir :

- `animation.ArtistAnimation` qui produit une animation à partir d'une série de données existentes.
- `animation.FuncAnimation` qui itère sur une fonction pour produire une animation en "live".

L'extrait de code 4.1 présente le fonctionnement de `animation.ArtistAnimation`. Cette méthode prend en arguments d'entrée **obligatoires** l'objet `figure` sur lequel construire l'animation et un objet de type **list** contenant les tracés qui la constitue (chaque tracé, qui est un objet de type `line`, doit lui même être dans un conteneur de type `list`). Pour utiliser cette méthode, il faut donc construire préalablement une liste de tous les tracés qui constituerons l'animation finale.

La fonction `animation.ArtistAnimation` accepte différents arguments **optionnels** comme par exemple l'argument `interval` qui fixe l'intervalle (en ms) entre chaque image. L'option `blit=True` permet quant à elle d'optimiser le processus de production de l'animation. Elle active un algorithme qui ne permet de changer que les éléments différents d'un tracé à l'autre au lieu de changer tout le tracé. La liste des options est disponible sur la page de manuel dédiée.

Notons qu'il est également possible de sauvegarder les animations. Le plus simple est généralement d'utiliser la méthode `save` héritée par les objets de type `animation` pour sauvegarder l'animation en fichier mp4.

```python
import matplotlib.pyplot as plt
import matplotlib.animation as animation

fig, ax = plt.subplots()     # Crée la figure et les axes
lines = []                   # Liste qui contiendra tous les tracés
for i in iterable:
    # Crée un objet 'line' contenu dans une liste : [line]
    line = ax.plot(MyData[i], animated=True)
    lines.append(line)       # et l'ajoute à 'lines'

ani = animation.ArtistAnimation(fig, lines, interval=50, blit=True)
plt.show()
```

Extrait de code 4.1 – ArtistAnimation

L'extrait de code 4.2 présente le fonctionnement de `animation.FuncAnimation`. Cette méthode prend en argument d'entrée **obligatoire** l'objet `figure` sur lequel construire l'animation et un objet de type `function` permettant de générer des tracés séquentiellement. La méthode `animation.FuncAnimation` accepte également des arguments **optionnels** qui sont listés sur la page de manuel dédiée.

```python
import matplotlib.pyplot as plt
import matplotlib.animation as animation

fig, ax = plt.subplots()     # Crée la figure et les axes
line = ax.plot([])           # Initialise le tracé

def animate(i):
    line.set_data(MyData[i]) # Met le tracé à jour avec les nouvelles données
    return line,             # Retourne le nouveau tracé

ani=animation.FuncAnimation(fig, animate, frames=50, interval=50, blit=True)
plt.show()
```

Extrait de code 4.2 – FuncAnimation

Sous-projet : 4.1 Une animation simple ⋆⋆

Il s'agit ici de créer une animation présentant l'évolution d'un sinus en fonction de son argument. La fonction considérée sera la fonction $f(t) = \sin(2\pi f t)$ sur l'intervalle $t \in [0, 1]$. L'animation fera apparaître l'évolution de la fonction $f(t)$ pour une fréquence f variant de 0 à 25 Hz.

Travail à réaliser

Écrire deux programmes :

1. Un programme affichant l'animation du sinus décrite ci-dessus à l'aide de la fonction `animation.ArtistAnimation`
2. Un programme affichant l'animation du sinus décrit ci-dessus à l'aide de la fonction `animation.FuncAnimation`

Sous-projet : 4.2 Une animation avancée ⋆ ⋆ ⋆

On se propose ici de créer une animation à partir des résultats obtenus au sous-projet 3.3. Plutôt que de superposer les formes d'ondes stationnaires et progressives à chaque instant t, il s'agit ici de construire une animation présentant l'évolution temporelle de chacun de ces deux types d'ondes.

Travail à réaliser

Utilisez la méthode `animation.ArtistAnimation` pour animer la figure 3.3. Pour ce cas particulier faisant intervenir plusieurs `axes` (subplots), souvenez vous bien que la méthode `plot` retourne un objet `line` contenu dans une liste et que `ArtistAnimation` prend en argument d'entrée une liste de listes et non pas une liste de listes de listes.

Projet 5

Les widgets avec `matplotlib`

Introduction

Le terme « widget » est un terme utilisé dans le jargon informatique pour désigner tous les éléments graphiques qui apparaissent sur une interface graphique : par exemple un bouton, une glissière, une case à cocher, un bouton radio, un menu déroulant, un sous-menu, une case de texte, une zone textuelle...

Ces widgets permettent d'interagir avec l'écran de l'ordinateur : soit avec votre doigt sur une tablette, avec la souris ou avec le clavier pour saisir du texte par exemple. Le module `matplotlib` intègre un petit nombre de ces éléments graphiques.

Le principe de fonctionnement est le suivant : un widget change d'état lorsque vous cliquez dessus, vous le survolez avec la souris, vous appuyez sur une touche de votre clavier, vous déplacez la glissière..., on dit que le widget reçoit un signal d'événement (« event » en anglais). Il est possible de relier une tâche à réaliser à chaque fois que l'état du widget change (c'est ce qu'on appelle une fonction « callback »). Cette fonction est alors exécutée et elle peut recevoir des informations de la part du widget (par exemple, position de la souris, lettre saisie au clavier, position de la glissière) sinon c'est à vous de récupérer vous-même ces informations.

Sous-projet : 5.1 Un widget simple ⋆⋆

Il existe dans le module `matplotlib` quelques « widgets » (ou éléments graphiques : glissière, bouton, bouton radio,...). Le programme ci-dessous (déposé sur UMtice) est un exemple très simple qui permet de modifier la fréquence d'un sinus à l'aide d'une simple glissière

```
"""
Slider and button Demo
======================
 Lisez bien les commentaires !
"""
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider, Button

fig, ax = plt.subplots()  # création du cadre qui contiendra le sinus
plt.subplots_adjust(bottom=0.25) # position de la ligne inférieure du cadre
                                 # 0=bas de la figure, 1=haut de la figure

# création du sinus
t = np.arange(0.0, 1.0, 0.001)
a0 = 5
f0 = 3
s = a0*np.sin(2*np.pi*f0*t)

# Tracé de la courbe et enregistrement dans "ligne" pour pouvoir la modifier
ligne, = plt.plot(t, s, lw=2, color='red')
ax.set_title('Sinus à la fréquence {:6.2f} Hz'.format(f0))
plt.axis([0, 1, -10, 10])

# création d'un cadre qui contiendra la glissière.
# les coordonnées [position coin inférieur gauche, longueur, hauteur]
# de la glissière avec position (0,0)= en bas à droite
# position (1,1)=en haut à droite de la figure
# longueur = 1 (100% de la largeur de la figure)
# hauteur = 1 (100% de la hauteur de la figure)
axfreq = plt.axes([0.15, 0.1, 0.75, 0.05], facecolor='lightgoldenrodyellow')
# création de la glissière dans ce cadre
sfreq = Slider(axfreq, 'Fréquence', 0.1, 30.0, valinit=f0)

# fonction qui sera appelée lorsqu'on déplace la glissière
def miseAjour(val):
    freq = sfreq.val        # on récupère la valeur de la glissière
    ligne.set_ydata(a0*np.sin(2*np.pi*freq*t)) # on met à jour la ligne
    ax.set_title('Sinus à la fréquence {:6.2f} Hz'.format(freq)) # le titre

```

```
41 # on attribue la fonction précédente
42 # à une modification de la valeur de la glissière
43 sfreq.on_changed(miseAjour)
44
45 # création d'un cadre qui contiendra le bouton 'Remise à zéro'
46 resetax = plt.axes([0.7, 0.025, 0.20, 0.04])
47 # création du bouton dans ce cadre
48 bouton = Button(resetax, 'Remise à zéro', color='red', hovercolor='lightblue')
49
50 def miseAzero(event):
51     sfreq.reset()  # réinitialisation de la glissière
52
53 # on attribue la fonction précédente à un clic sur le bouton
54 bouton.on_clicked(miseAzero)
55
56 plt.show()
```

Travail à réaliser

- Récupérer le programme sur UMtice et lancer-le pour comprendre ce que réalise ce programme.
- Adapter le programme ci-dessus afin de tracer la courbe **sous** la glissière, de mettre le bouton de réinitialisation en haut à gauche.
- Ajouter 2 autres glissières afin de modifier l'amplitude et la phase du sinus.

Sous-projet : 5.2 Un widget avancé ⋆⋆⋆

À VENIR !!!

Projet 6

I *Lecture de fichiers*

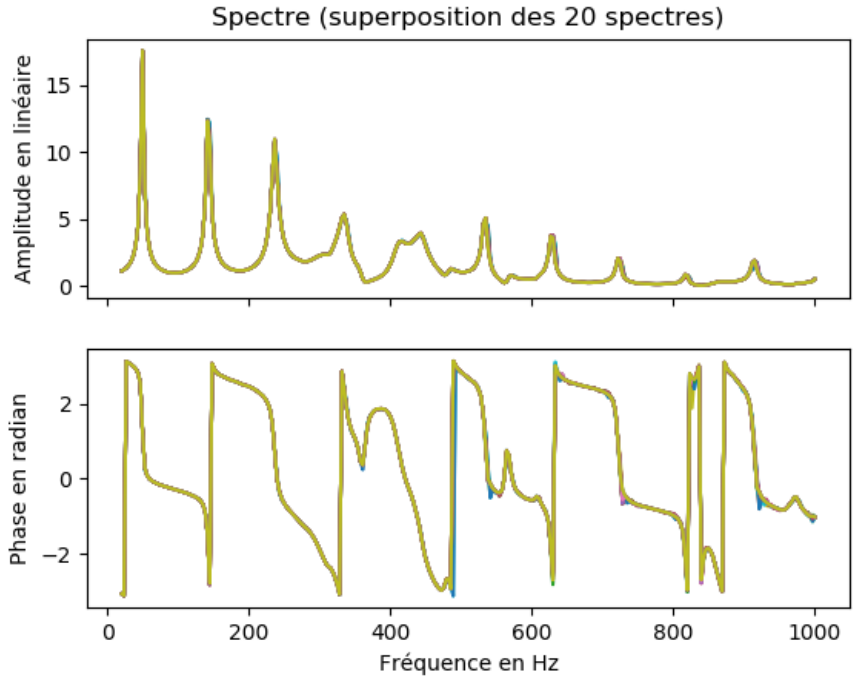
En acoustique, il n'est pas rare qu'au cours de vos mesures, vous obteniez une grosse quantité de fichiers qu'il vous faut ensuite analyser afin d'en extraire l'information recherchée. Cette information peut être souvent obtenue visuellement à partir de courbes que vous tracez mais également numériquement à partir d'un calcul effectué sur vos données (par exemple, énergie d'un signal).

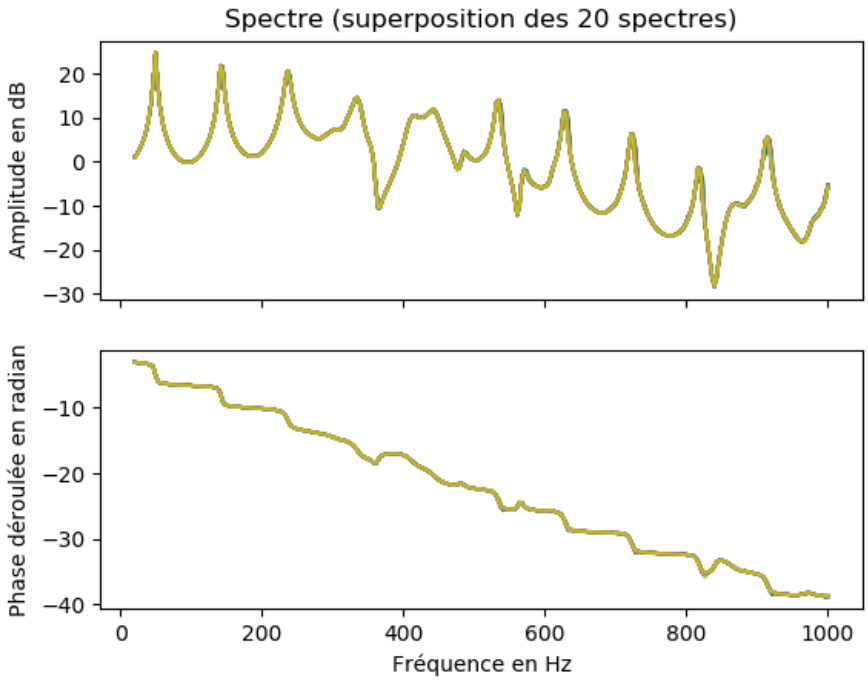
Dans le cas d'une extraction visuelle de l'information, celle-ci n'est possible que si le nombre de courbes n'est pas trop important (analyse dite manuelle). Lorsque le nombre de courbes (ou de fichiers) devient trop grand, il faut recourir de toute façon à une analyse dit automatique à l'aide d'un programme adapté (par exemple, recherche de la fréquence du pic d'amplitude la plus grande sur des milliers de courbes).

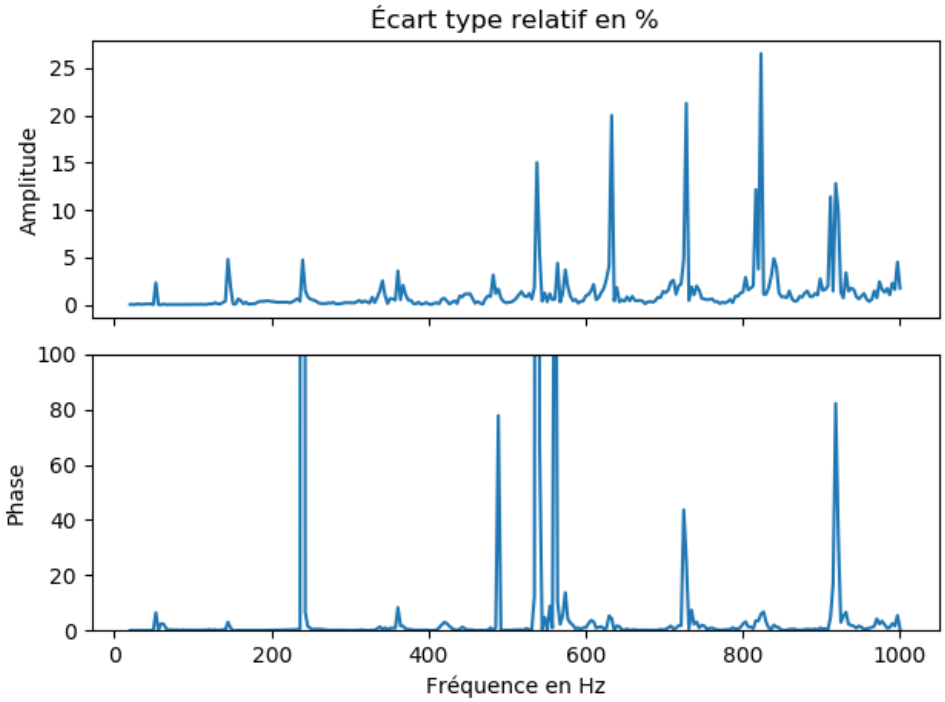
Le but des exercices suivants est de vous faire réfléchir sur une manière de traiter l'information contenu dans un nombre N de fichiers enregistrés sous des formats divers et variés. Un certain nombre (limité) de fichiers vous sont donc fournis sur UMTice mais vos programmes devraient fonctionner même si ce nombre devenait très très grand.

Sous-projet : 6.1 Test de répétabilité : statistiques simples ⋆⋆

Dans cet exemple, il s'agit d'un test de répétabilité d'une mesure de spectre dans un tube (la position du microphone, de la source et le tube étant inchangés). Cette mesure est répétée un certain nombre de fois et l'objectif est d'écrire un programme permettant de reproduire les trois courbes suivantes permettant de se rendre compte que la mesure d'une résonance n'est pas aussi facile qu'on le croit !


Spectre (superposition des 20 spectres)
Amplitude en linéaire
15
10
5
0
Phase en radian
2
0
−2
0
200
400
600
800
1000
Fréquence en Hz


Spectre (superposition des 20 spectres)
Amplitude en dB
20
10
0
−10
−20
−30
Phase déroulée en radian
−10
−20
−30
−40
0
200
400
600
800
1000
Fréquence en Hz



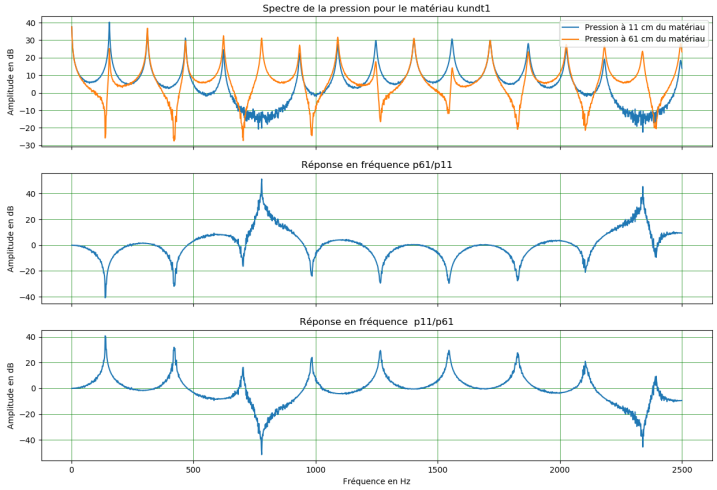
Sous-projet : 6.2 Analyse de données expérimentales : tube de Kundt ⋆⋆⋆

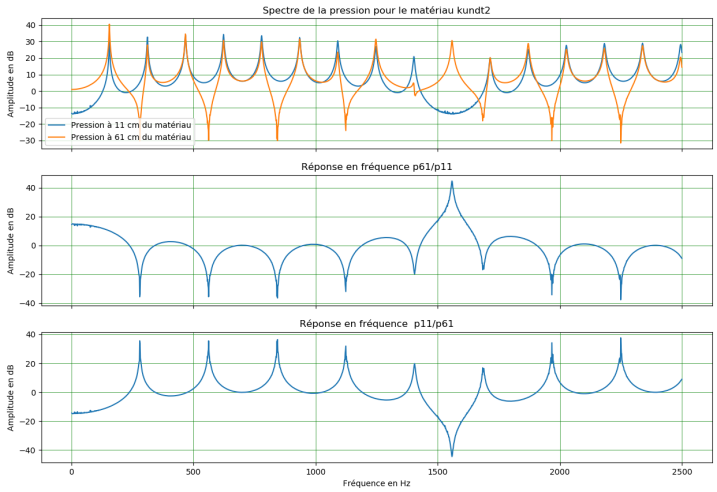
Un étudiant a effectué la mesure de la pression acoustique dans un tube de Kundt de 1.1 mètre de long à l'aide d'un microphone, déplacé dans le tubeà l'aide d'un robot. Les mesures sont effectuées tous les 2 cm et commencent à 1 cm de la surface du matériau dont on veut déterminer le coefficient de réflexion. L'expérience est répétée pour trois matériaux différents donc trois coefficients de réflexion différents.
Les données de chaque expérience sont rangées dans 3 répertoires différents `kundt1`, `kundt2`, `kundt3`. Dans chaque répertoire, il y a 50 fichiers numérotés de 1 à 50. Le numéro indique l'emplacement de la mesure : mesure n° 1 à 1 cm du matériau, mesure n° 2 à 3 cm, mesure n° 3 à 5 cm...

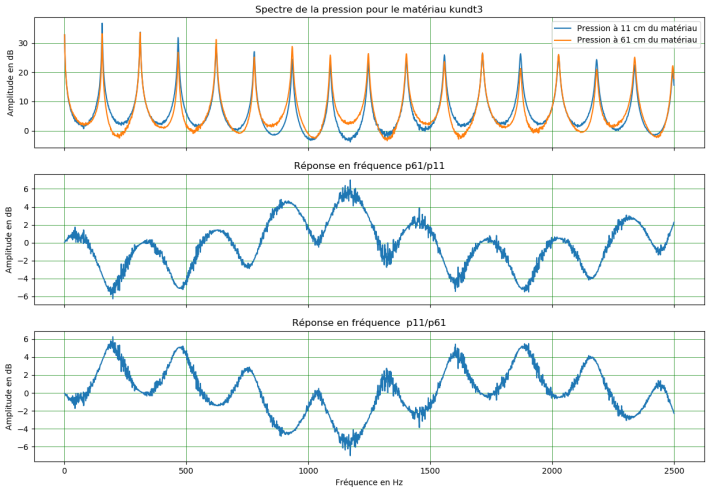
Chaque fichier contient un spectre. Dans chaque fichier, on peut y voir 2 colonnes de nombres : la première colonne représente la partie réelle de spectre, la deuxième représente la partie imaginaire. Chacune des lignes correspond à la fréquence étudiée. Le pas fréquentiel vaut de 5000/4096 Hz sachant que la première ligne correspond à la fréquence 0 Hz.

Travail à réaliser

1. Écrire un programme qui superpose les spectres de pression en dB, mesurés pour deux positions de microphones (à 11 cm (p_{11}) et à 61 cm (p_{61}) du matériau, par exemple) puis, sur les deux figures en dessous, la réponse en fréquence en dB entre les deux microphones : p_{11}/p_{61} et p_{61}/p_{11}. Utiliser ce programme pour les 3 matériaux, enregistrer les courbes obtenues et les comparer ensuite. Dans le cas `kundt1` :
   - Justifier l'écart fréquentiel entre les pics de résonance des spectres.
   - À quoi correspond, selon vous les zones bruitées du spectre sur la courbe de la pression p_{11}. Justifiez-le par un calcul simple.
   - À quoi correspond l'écart fréquentiel entre les pics des réponses en fréquence. Justifiez-le par un calcul simple.





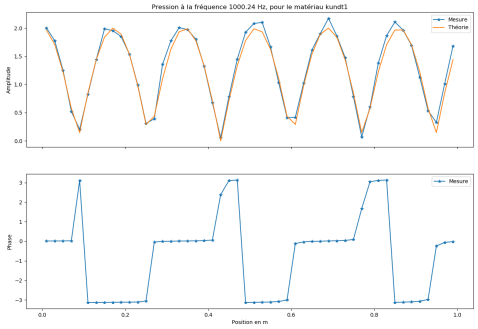


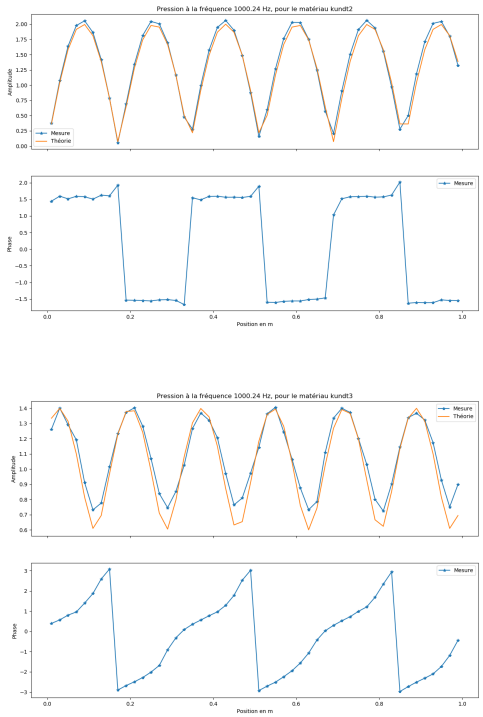
2. Écrire un programme qui permet d'afficher dans la même figure, l'un en dessous de l'autre, le module et la phase de la pression mesurée $p_{\text{mesurée}}(x)$ en fonction de la position x dans le tube à une fréquence choisie par l'utilisateur (le titre de la courbe doit indiquer la **fréquence réelle** du signal, c.-à-d., la plus proche de celle choisie par l'utilisateur).

   Faire ensuite un ajustement manuel de la courbe afin d'obtenir une valeur approchée de module de coefficient de réflexion $|R|$ et la phase ϕ du coefficient de réflexion de chaque matériau. Pour cela, on utilisera l'équation théorique suivante :

$$p_{\text{théorique}}(x) = e^{jkx} + |R|e^{j\phi}e^{-jkx}$$

   où $k = \omega/c_0$ avec $c_0 = 344$ m/s, on superposera sur une même courbe $p_{\text{mesurée}}(x)$ et $p_{\text{théorique}}(x)$ puis modifiera les valeurs de R et ϕ pour que les courbes coïncident à peu près.
   Quelle est l'influence de $|R|$ et de ϕ sur la courbe théorique ?





Sous-projet : 6.3 Lecture de fichiers audio sous Python ⋆⋆⋆

Les objectifs de ce sous-projet sont :

- de savoir lire des fichiers audio (ou pas) du monde réel,
- de savoir en extraire une partie donnée en vue d'analyses ultérieures.

Les fichiers sonores sont très souvent enregistrés au format `.wav`. Il est donc important de savoir les lire sur le disque dur, d'en afficher les contenus correctement dans le domaine temporel, de savoir les manipuler dans le domaine temporel et de les enregistrer à nouveau éventuellement sur le disque dur dans le même format ou pas. Ces exercices vous serviront l'année prochaine en Traitement du Signal Numérique. **Conservez-les précieusement !**

Utiliser les fonctions du sous-sous-module `wavfile` du module `scipy` :

```
from scipy.io import wavfile
```

Exercices

Chaque exercice correspondra à un programme que nous vous conseillons d'archiver très soigneusement sur votre espace personnel `H:`, il est plus que probable que ces morceaux de code vous serviront plus tard. N'hésitez pas à bien commenter ces codes car ils vous serviront de brique élémentaire par la suite.

1. Écoutez au casque le fichier `countdown1.wav` puis lisez-le à l'aide de Python, affi-

chez à l'écran la valeur de la fréquence d'échantillonnage ainsi que le nombre de points du fichier puis tracez le signal après avoir créé un axe temporel correspondant à la durée du signal.

2. Lisez les fichiers `countdown1.wav` ET `countdown2.wav` et **superposez** les deux signaux sur une même figure après avoir créé un axe temporel correspondant à la durée des signaux.
3. Lisez le fichier `countdown2.wav` et **extrayez** le son correspondant au mot « seven », tracez le signal correspondant sur une figure puis enregistrez ce mot dans un fichier appelé `seven.wav`.
4. Lisez le fichier `countdown1.wav` et **extrayez** le son correspondant au mot « five », tracez le signal correspondant puis enregistrez ce mot répété cinq fois de suite dans un fichier appelé `fivefive.wav`.
5. Lisez le fichier `countdown2.wav` et extrayez le son correspondant au mot « seven » mais cette fois-ci on souhaite **conserver la durée totale** du signal initial [1], tracez le signal correspondant puis enregistrez ce mot dans un fichier appelé `riensevenrien.wav`.
6. Lisez le fichier `countdown1.wav` et enregistrez-le dans un fichier `.wav` en modifiant la fréquence d'échantillonnage de la manière suivante : $F_s \times 1.33$ (pour le fichier `faster.wav`) et $F_s \times 0.66$ (pour le fichier `slower.wav`) puis écoutez-les.
7. Lisez le fichier `countdown1.wav` et enregistrez le signal au format `numpy` binaire (extension `.npy` donc `countdown1.npy`) à l'aide de la commande `np.save` puis au format texte (`countdown1.txt`) à l'aide de la commande `np.savetxt`. Comparez avec la taille du fichier `.wav` initial. Quels sont les avantages ou intérêts des 3 types de format de fichiers.
8. Lisez le fichier `chirp.wav` et **tracez-le**. Isolez le chirp du milieu (le deuxième) puis enregistrez-le à l'envers (retournement temporel) dans le fichier `prihc.wav`. Écoutez ce fichier.
9. Lisez le fichier `gong.wav` et **tracez-le**. Enregistrez le signal à l'envers dans le fichier `gnog.wav`. Écoutez ce fichier. Faites de même avec le fichier `countdown1.wav` en nommant le fichier `countup1.wav`.
10. Lisez le fichier `moteurdiesel.wav` puis tracez le signal après avoir créé un axe temporel correspondant à la durée du signal (attention, ce fichier est un fichier stéréo). Enregistrez la voie droite et la voie gauche dans des fichiers séparés : `dieseldroite.wav`, `dieselgauche.wav`.
11. Il n'est pas rare que des logiciels de mesure vous fournissent une réponse en fréquence dans des fichiers texte. Très souvent, il y a une ligne d'entête qui décrit le contenu du fichier (à quoi correspondent les colonnes du fichier). Le but de cet exercice est de lire le fichier `FRF.txt` et d'afficher la courbe du module de la réponse en fréquence en fonction de la fréquence.

1. mettre à zéro le signal lorsque ce n'est pas le mot « seven »

12. (DIFFICILE) Lisez les fichiers `countdown1.wav` ET `countdown2.wav` et enregistrez ces deux signaux dans un même fichier `.wav` en mettant sur la voie droite le premier signal et sur la voie gauche le deuxième.

Projet 7

Diagramme de directivité

Les représentations graphiques peuvent se faire en coordonnées cartésiennes mais aussi en coordonnées polaires.

La fonction `plt.polar` permet de représenter des fonctions en coordonnées polaires. Ceci présente un grand intérêt pour les diagrammes de directivité. En réalité, cette fonction ne fait que de changer les coordonnées en coordonnées polaires puis appelle la fonction `plt.plot`.

Afin d'avoir plus de fonctionnalités de configuration, il faut donc utiliser les commandes

```
ax=plt.subplot(111,polar=True)
plt.plot(...)
```

plutôt que `plt.polar` moins configurable.

Petit cours d'électroacoustique :

Il existe différents types de diagramme de directivité « classique » pour les microphones :

Omnidirectionnel $\rho = 1$

Bidirectionnel $\rho = \cos\theta$

Cardioïde $\rho = 1 + \cos\theta$

Supercardioïde $\rho = 1 + 3\cos\theta$, par exemple

Hypercardioïde $\rho = 1 + 2\cos\theta$, par exemple

où $|\rho|$ désigne l'amplitude et θ l'angle variant entre 0 à 2π.

Sous-projet : 7.1 Tracé de diagramme de directivité ⋆⋆

Écrire un programme affichant un menu (exemple ci-dessous)

```
Choisissez un diagramme de directivité dans la liste ci-dessous :
------------------------------
1 -> omnidirectionnel
2 -> bidirectionnel
3 -> cardioide
4 -> hypercardioide
5 -> supercardioide
------------------------------
Indiquez le numéro du diagramme souhaité :
```

permettant à l'utilisateur de choisir le type de diagramme à afficher. Le programme trace ensuite l'UNE des courbes suivantes (vous aurez besoin de la fonction `plt.fill_between`) :