Université Aix-Marseille 2018-2019
Master Traitement du signal et des images 1ère année

Mathématiques pour le signal et l'image

TP1 : bases de Fourier et filtrage.

Tous les fichiers à télécharger dont vous avez besoin sont dans la rubrique TP1 du site Ametice.

1 Objectifs

1. Apprendre à utiliser l'outil numérique qui permet de calculer des coefficients dans la base de Fourier.
   Cet outil s'appelle la transformée de Fourier discrète.
2. Restaurer un signal dégradé en le filtrant passe-bas, c'est à dire en mettant à 0 les hautes fréquences d'un signal donné. On comprendra ce qu'on obtient en pratique à l'aide de ce qui a été vu en cours.

2 Recommandations diverses

— Avant de commencer à travailler on vous recommande d'ouvrir dans le menu principal en haut à gauche les préférences du navigateur Web que vous utilisez. Dans la rubrique Fichiers et applications/Téléchargements choisissez « Toujours demander où enregistrer les fichiers ».
— Dans tout ce qui suit vous êtes libre de travailler avec l'interface Python que vous préférez. On suggère pour tous ceux qui n'ont pas de préférence particulière de travailler avec Spyder que l'on peut lancer avec l'application Anaconda.
— Dans tous les cas on vous conseille de travailler avec l'éditeur de texte que vous souhaitez sur un fichier procédure `tp1.py` que vous enregistrez dans le dossier dans lequel vous allez travailler. Vous pouvez créer bien sûr autant de fichiers procédures que vous le souhaitez. Cela vous permet de garder une trace de ce que vous aurez fait.

  Dans le cas où cela vous paraît adapté vous pouvez aussi créer les fonctions qui vous semblent adéquates.
— On vous recommande de configurer aussi les préférences de Spyder en particulier pour les figures. Ainsi ouvrez les préférences dans le menu principal **Python** en haut à gauche, et dans la rubrique **Console Ipython**, **graphics** cochez l'option **Backend : Automatic** afin que vous puissiez manipuler plus facilement les figures.
— Commentez ou expliquez vos codes à l'aide du signe #.

3 Numérisation d'un signal

Un signal de la variable continue, c'est à dire une fonction f de la variable continue, doit être numérisé pour pouvoir être traité par un ordinateur.

On choisit l'intervalle de temps que l'on notera ici $[0, T]$ (sans perte de généralité), sur lequel on veut travailler. Puis on choisit un intervalle de temps $t_s = T/N$ (N entier), qui indique à quelle cadence on va prendre les échantillons. On note $f_s = \frac{1}{t_s}$ qui est appelée « **fréquence d'échantillonnage** ».

Le signal numérisé est alors obtenu en calculant pour $0 \leq n \leq N-1$ $u[n] = f(nt_s) = f\left(\frac{n}{f_s}\right) = f\left(\frac{nT}{N}\right)$. Le vecteur $u = (u[0], \dots u[N-1]) = \left(f\left(\frac{0}{N}\right), \dots, f\left(\frac{(N-1)T}{N}\right)\right)$ correspond donc à la

numérisation du signal f (voir figure 1).

On se rend compte sur cette figure 1 que si la fréquence d'échantillonnage f_s est trop petite, le signal pourrait être dégradé au moment de la numérisation.

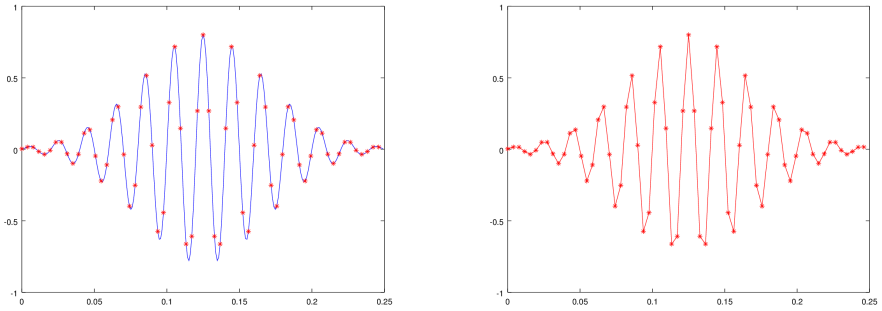
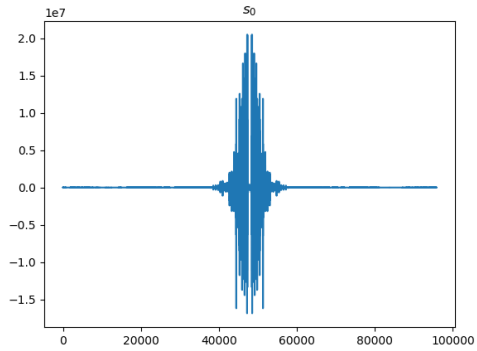


FIGURE 1 – A gauche : le signal original f représenté sur $[0, T]$ en bleu a été échantillonné et on a donc construit le vecteur $u = (u[0], \dots u[N-1])$ dont les valeurs correspondent aux ordonnées des points rouges. A droite : on trace en les reliant par des segments de droites les points $(t_n, u[n])$ avec $t_n = \frac{nT}{N}$.

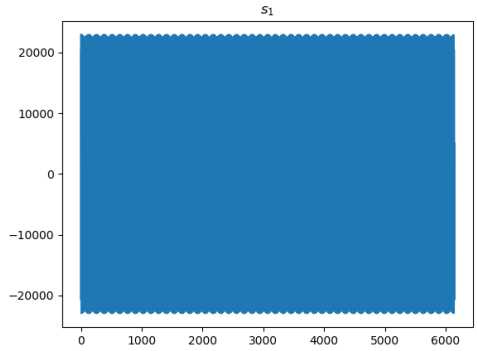
4 Travaux pratiques

Exercice 1 (*Quizz des signaux*)

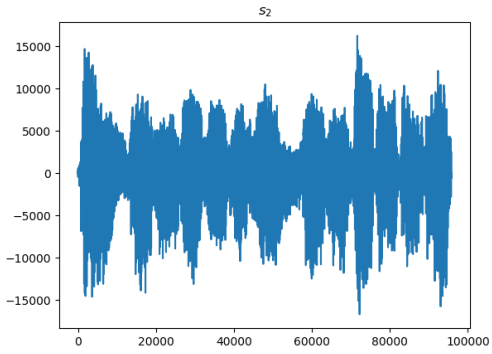
On considère quatre signaux s_0, s_1, s_2, s_4 : deux d'entre eux sont les parties réelles des transformées de fourier discrètes des deux autres, qui sont quant à eux des signaux réels et ont été obtenus à partir des sons `son1.wav` *et* `son2.wav`*. Indiquer qui est la transformée de Fourier discrète de qui en justifiant vos affirmations.*
On peut s'aider de la section suivante qui détaille les commandes Python qu'on peut utiliser et on pourra écouter les morceaux de ces fichiers avec le lecteur de votre choix (par exemple VLC fonctionne très bien).



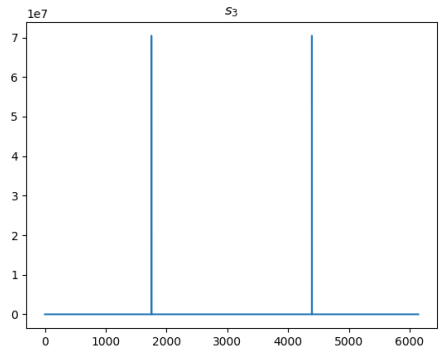
s_0



s_1



s_2



s_3

Exercice 2 (*Sinusoïdes pures*)

1. *Créer sans boucle le vecteur u de taille $N = 2048$, tel que pour $n = 0, .., N-1$, $\omega_0 = 100$ et $\omega_1 = 102$ (ici $f_s = N$), on a*

$$u[n] = \cos(2\pi\omega_0 * n/N) + \cos(2\pi\omega_1 * n/N).$$

2. *Le visualiser à l'aide des fonctions du module* `matplotlib.pyplot`.
3. *Calculer $\hat{u}$ à l'aide de la fonction* `fft` *du module* `numpy.fft`.
4. *Dans tous les cas suivants on se demandera si les résultats obtenus sont cohérents avec ce que les mathématiques nous prédisent. Visualiser le vecteur dont les coordonnées*

sont les suivantes.

(a) *les parties réelles des* $\hat{u}[k]$ *pour* $k = 0, .., N-1$.

(b) *les parties imaginaires des* $\hat{u}[k]$ *pour* $k = 0, .., N-1$.

(c) *les parties réelles des* $\hat{u}[k]$ *en fonction de* $k = -N/2, \ldots, N/2-1$.

(d) *les modules des* $\hat{u}[k]$ *en fonction de* $k = -N/2, \ldots, N/2-1$.

5. *Peut-on vérifier sur les figures obtenues qu'il y a deux fréquences principales d'oscillation dans le signal et les identifier ?*

Exercice 3

Le son `son3.wav` *est le résultat de la dégradation d'un signal de parole. On veut le restaurer.*

1. *Transformer le signal en un vecteur* x*, le visualiser.*
2. *Tracer le module de sa transformée de Fourier discrète* $\hat{x}$*.*
   Cela revient à changer de base : on calcule les coefficients de x *dans la base de Fourier.*
3. *On se propose d'effectuer un filtrage linéaire pour restaurer le signal original à l'aide du filtre étudié dans le TD 4. La fonction* `filtrage` *permet de calculer* $y = h \star x$ *où* h *est tel que* $\hat{h}$ *vérifie*

$$\hat{h}[k] = \begin{cases} 1 & si \quad -k_c \leq k \leq k_c\,, \\ 0 & sinon. \end{cases}$$

   Ici k_c *est proportionnel à la fréquence « de coupure » de* ω_c *qui est une fréquence physique.*

   À l'aide de la fonction `filtrage` *(a télécharger sur Ametice ou disponible sur l'annexe ci dessous) qui permet de calculer* $y = h * x$*, calculer une estimation du signal d'origine où la fréquence parasite est éliminée voire aténuée.*

5 Commandes Python

5.1 Commandes utiles pour l'exercice 1

Les fichiers de données que nous allons considérer sont des fichiers au format **.wav** comme les fichiers **son1.wav** et **son2.wav**.

Il faut pouvoir transformer ces fichiers en un vecteur dont les composantes sont les valeurs du signal aux instants nT/N, $n = 0, \ldots, N-1$. Pour cela, on utilise le module `scipy.io.wavfile` et sa commande `read`.

— On peut exporter les informations du fichier `.wav` à l'aide des commandes suivantes.

```
import scipy.io.wavfile as wavfile

fs, u = wavfile.read('son1.wav')
```

Ces commandes nous permettent d'obtenir le vecteur `u` recherché et la fréquence d'échantillonnage `fs` avec laquelle le son a été numérisé, c'est-à-dire combien de valeurs par seconde ont été recueillies.

— Pour tracer le vecteur on peut utiliser les commandes du module `matplotlib`

```
import matplotlib.pyplot as plt

plt.figure()
plt.plot(u)
plt.show()
```

— Si on veut tracer le vecteur en fonction du temps il faut alors utiliser l'information sur la fréquence d'échantillonnage

```
N=u.size
T=N/fs

t=np.arange(0,T,1/fs)
plt.figure()
plt.plot(t,u)
plt.show()
```

— La meilleure façon de calculer la transformée de Fourier discrète d'un vecteur `u` est l'algorithme de transformée de Fourier rapide (en anglais FFT pour Fast Fourier Transform) qui fonctionne en $\mathcal{O}(N\log(N))$ opérations. C'est un des algorithmes les plus rapides actuellement existants. Plusieurs librairies de Python ont implémenté cet algorithme. Nous nous proposons d'utiliser ici le module `numpy.fft` de la librairie `numpy`.

La fonction de cette librairie que nous pouvons utiliser pour calculer la transformée de Fourier discrète d'un vecteur u est ainsi `numpy.fft.fft`.

```
import numpy as np

uchap=np.fft.fft(u)
```

— Si on veut visualiser la partie réelle du vecteur *uchap* on effectue

```
uchapre=uchap.real
```

— Si on veut visualiser le module du vecteur *uchap* on calcule

```
uchapm=np.abs(uchap)
```

— L'intérêt de la transformée de Fourier est de pouvoir analyser les fréquences présentes dans le signal.
En particulier la propriété de symétrie hermitienne dans le cas d'un signal u réel

$$\overline{\hat{u}[k]} = \hat{u}[N-k]$$

nous invite à regarder la transformée de Fourier discrète d'un signal en utilisant $(\hat{u}[-N/2], \hat{u}[-N/2+1], \ldots, \hat{u}[-1], \hat{u}[0], \hat{u}[1] \ldots, \hat{u}[N/2-1])$ afin d'avoir les fréquences bien centrées en 0 et pour faciliter l'interprétation.

Cela se fait à l'aide de la commande `fft.fftshift` du module `numpy.fft`

```
plt.figure()
plt.plot(np.fft.fftshift(uchapre))
plt.show()
```

— Dans le cas des sons on va renormaliser l'axe des abscisses pour pouvoir mesurer des fréquences physiques, qui sont mesurées entre $-f_s/2$ et $f_s/2$. Ainsi on écrit

```
N=uchapre.size

freq=np.arange(-(N//2),N-(N//2))*fs/N
plt.figure()
plt.plot(freq,np.fft.fftshift(uchapre))
plt.show()
```

5.2 Commandes Python utiles pour l'exercice 2

— Le module `numpy` permet de créer facilement des vecteurs. La fonction `np.arange` permet de créer un vecteur dont les coordonnées sont régulièrement espacées. Par exemple

```
import numpy as np

u=np.arange(1,10)
```

crée le vecteur $u = (1, 2, 3, \ldots, 9)$.

— Si on veut un écart non entier entre les coordonnées on peut écrire

```
import numpy as np

u=np.arange(1,2,0.1)
```

ce qui crée le vecteur $u = (1, 1.1, 1.2, \ldots 1.9)$.

— Quand on veut appliquer une fonction f classique définie sur $\mathbb{R}$ (comme la fonction sinus, cosinus, puissance ...) et calculer $(f(x_0), f(x_1), \ldots f(x_{N-1}))$ on peut tout simplement calculer $u = (x_0, \ldots x_{N-1})$ et appliquer f sur u. Cela donne par exemple

```
import numpy as np

v=np.pi*u
y=np.sin(v)
```

qui calcule ici $y = (sin(\pi * 1), sin(\pi * 1.1), sin(\pi * 1.2), \ldots sin(\pi * 1.9))$.

— Quand on veut tracer $(f(x_0), f(x_1), \ldots f(x_{N-1}))$ en fonction de $(x_0, \ldots x_{N-1})$ on va effectuer les commandes suivantes

```
import numpy as np

t=np.arange(0,2,0.1)
y=np.sin(np.pi*t)
plt.figure()
plt.plot(t,y)
plt.show()
```

qui trace donc les couples de points (t_i, y_i) (ici $y_i = \sin(\pi t_i)$ pour $i = N$ la taille de t qui est aussi celle de y.

5.3 Commandes Python utiles pour l'exercice 3

1. Dans le cas des sons on renormalise l'axe des abscisses pour pouvoir mesurer des fréquences physiques, qui sont mesurées entre $-f_s/2$ et $f_s/2$. Ainsi on écrit

```
N=uchapre.size

freq=np.arange(-(N//2),N-(N//2))*fs/N
plt.figure()
plt.plot(freq,np.fft.fftshift(uchapre))
plt.show()
```

2. Un vecteur u peut être transformé en son et stocké dans le fichier `newson.wav` par exemple à l'aide de la commande `write` du module `scipy.io.wavfile`.

   Cependant nous avons intérêt à ce que les valeurs codées dans le fichier `.wav` ne soient pas trop grandes pour que les lecteurs standards puissent lire facilement le son. Nous ferons donc systématiquement la normalisation suivante, qui consiste à commencer par mettre toutes les coordonnées de u entre 0 et 1

```
import scipy.io.wavfile as wavfile
```

```
m=np.max(np.abs(u))
u=u/(1.1*m)
wavfile.write('newson.wav',fs,u)
```

6 Compte-rendu du TP

Un compte-rendu pour les deux séances de TP sera à rendre par groupe de deux maximum (avec mention sur les documents des membres du groupe) sur Ametice.

Annexe

```
def filtrage(omegac,x,fs):
    xchap=np.fft.fft(x)
    N=xchap.size
    M=int(np.floor(N*omegac/fs))
    mask=np.ones(N)
    coord=np.arange(M+1,N-M,1)
    mask[coord]=0
    xfiltchap=mask*xchap
    xfilt=np.fft.ifft(xfiltchap)
    xfilt=xfilt.real
    return xfilt
```