TP2 : Débruitage de la parole (Fourier, Soustraction spectrale)

Le travail à rendre contiendra le fichier .py, et l'analyse du débruitage (.odt, .doc ou .pdf), le tout compressé dans un fichier .zip. voir fin de l'énoncé
Il est possible de se mettre par binôme, mais constitué lors des séances

[Séance en « salle » machine : ~3h]

Objectifs : Réaliser une analyse OLA d'un signal de parole par Fourier (fenêtrage, spectre d'amplitude, spectre de phase, reconstruction du signal) et réaliser un algorithme de débruitage sur le principe de la soustraction spectrale.

Outils : Python et Winsnoori pour lire et visualiser le signal modifié.

Le but du TP est de réaliser un traitement sur le signal de parole dans le domaine fréquentiel. L'entrée est un signal de parole, la sortie doit être un signal de parole.
Au fur et à mesure du TP, nous ajouterons des étapes de traitement entre l'entrée et la sortie.
Pour tester le programme avant les deux dernières étapes (8. Estimation du bruit et 9. Débruitage par soustraction spectrale), vous utiliserez le fichier test_seg.wav. Ensuite, pour l'estimation du bruit et le débruitage, vous utiliserez les fichiers *_bruit_ndB.wav avec n le niveau signal sur bruit (SNR – signal noise ratio) en décibels.

1. Récupération du signal

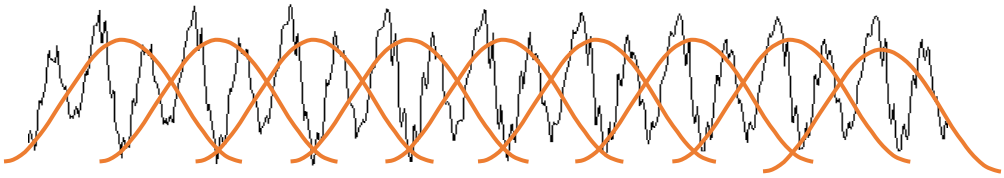
Ouvrez un fichier wav (et affichez l'information de la fréquence d'échantillonnage, et la taille du fichier en échantillons et en millisecondes)

Pour lire un fichier wav : **from scipy.io.wavfile import read** La fonction read renvoie la fréquence d'échantillonnage et le tableau des valeurs du signal

*[Rappel : si le signal est échantillonné à 16000 Hz : 4ms correspond à 4*16000/1000 échantillons]*

2. Boucle OLA (*OverLap and Add*)

Réalisez une boucle qui récupère le signal tous les 8 ms par morceau de 32ms (on se prépare à faire un traitement sur ces morceaux). Vous placerez les échantillons dans un tableau temporaire (donc de taille équivalent à 32 ms).



Réalisez une fonction qui renvoie une fenêtre de Hamming.
def fenetrageHamming(size) Avec size la taille de la fenêtre
(formule Hamming : voir poly s.18)

Puis réalisez le fenêtrage du morceau de signal (de 32 ms) avec la formule pour le fenêtrage : $signal\_fen[i] = signal\_extrait[i] \times \text{hamming}[i]$
pour i entre 0 et size.

Enfin, nous allons reconstruire le signal :
Dans un nouveau tableau (de la taille du signal de départ) signal_modif, replacez au même endroit les morceaux (voir la structure du programme ci-dessous).
Comme les morceaux sont fenêtrés, il faut « enlever » le poids de ce fenêtrage. Ainsi, au fur-et-à-mesure de la boucle, construisez un nouveau tableau de même taille que le signal (ex : somme_hamming) en y ajoutant des fenêtres de Hamming. Ici, le recouvrement est constant et fait de telle manière que la valeur du tableau sera constante (sauf aux extrémités) [ici 2.16].
Puis au final (après la boucle) divisez signal_modif par somme_hamming .

Enfin écrivez-le dans un .wav. Si tout se passe bien, le signal final n'a pas été modifié.

Pour écrire dans un fichier wav : **from scipy.io.wavfile import write**
write("resultat.wav", fs, np.int16(*signal_modif*))
(*)Pour la reconstruction, il faut *caster* les échantillons en np.int16 avant de les écrire dans le fichier Wav (puis que le signal est quantifié en 16 bits)

Votre programme aura donc la structure suivante :

Pour i de 0 à taille_signal-n de *m en m*
- ***Récupération de la fenêtre à l'instant i et de taille m (2)***
- Fenetrage Hamming (2.)
- Fourier (3.)
  - Spectre d'amplitude (4.)
  - Spectre de phase (6.)
- Traitement sur le spectre d'amplitude (8. 9.)
- Reconstruction du spectre (7.)
- Fourier inverse (3.)
- ***Reconstruction du signal (2.) : ajout de la fenetre au signal final***

Fin pour
- ***Reconstruction du signal (2.) : division par la somme des fenêtres de Hamming***

Les étapes 2., et 3. sont des opérations « symétriques », elles s'opèrent en début et en fin de boucle (seul la division finale est en dehors de la boucle).

3. Transformée de Fourier

Sur chaque fenêtre de signal, nous allons maintenant calculer la **transformée de Fourier**. Nous utiliserons pour cela, la fonction fft de numpy.

import numpy.fft as FFT FFT.fft(signal_modif, fftsize) Renvoie un tableau de taille fftsize de valeurs complexes

Réaliser la FFT sur 1024 points.
Comme le signal est une fenêtre de 32 ms (~705 échantillons), normalement il faudrait compléter par des zéros pour atteindre 1024 échantillons. (ou ce qui revient à initialiser le tableau avec des zeros), mais la fonction fft le fait elle-même.

Puis ensuite sa transformée inverse.

FFT.ifft(spectre, fftsize) Renvoie un tableau de taille fftsize de valeurs complexes, il ne faut garder que la partie réelle : np.real(…).

On doit réobtenir le signal original.

4. Spectre d'amplitude

Réalisez une fonction qui calcule le **spectre d'amplitude** (sur une fenêtre) c'est-à-dire le module du nombre complexe :

def spectreamplitude(spectre, fftsize) ;

Il faut calculer le module du spectre :

$$Spectre\_amplitude[k] = |X_k(\omega)| = \sqrt{\Re(X_k)^2 + \Im(X_k)^2}$$

Mais il suffit d'utiliser directement np.abs(…), qui va calculer directement le module du nombre complexe.
Remarque : on pourrait ne calculer que la moitié du spectre, puisque la partie réelle est symétrique et la partie imaginaire est asymétrique (donc le spectre d'amplitude est symétrique).

Rappel : Pour un spectrogramme (la représentation du son par le spectre d'amplitude) on calcule normalement le spectre d'amplitude en prenant le logarithme (multiplié par 20) du module, cela revient à :

$$Spectre\_amplitude[k] = 20.\log(|X_k|)$$

Stocker le spectre d'amplitude (avec le logarithme) ce qui nous servira pour afficher le spectrogramme.
(mais gardez une version du spectre d'amplitude sans logarithme pour faire les calculs du 5.)

5. (Pause sur le débruitage)

Après la boucle OLA, affichez les spectres d'amplitudes (avec le logarithme) mais seulement de 0 à fftsize/2, puis que le spectre d'amplitudes est symétrique.

Pour aficher vous pouvez utiliser la fonction imshow(…) de la librairie Matlibplot. Il faudra transposer le tableau de stockage des spectres (.transpose()).

Pour l'affichage, aspect=auto

6. Spectre de phase

[retour dans le débruitage] Un nombre complexe est défini par son module et son angle. Comme il faudra reconstituer le spectre (complexe) après modification du spectre d'amplitude, il faut garder en mémoire le spectre de phase (l'angle).
Réalisez une fonction qui calcule le **spectre de phase** :

def spectrephase(spectre, fftsize) ;

avec la phase :

$Spectre\_phase[k] = \varphi_k = Arctan\left(\frac{\Im(X_k)}{\Re(X_k)}\right)$ (c'est-à-dire l'angle du complexe X_k)

Là encore numpy nous facilite le travail : np.angle(…) d'un nombre complexe et nous obtenons la phase.
Remarque : on pourrait ne calculer que la moitié du spectre, puisque la partie réelle est symétrique (dite aussi *paire*) et la partie imaginaire est *impaire* (donc le spectre de phase est *impaire*).

7. Reconstruction du spectre

Reconstruire un spectre complexe à partir du module et de la phase :

def spectrereconstruction (spectre_amplitude, spectre_phase,fftsize)

avec la formule :

$$X_k(\omega) = \Re(X_k) + j\Im(X_k) = |X_k(\omega)|\cos(\varphi_k) + j|X_k(\omega)|\sin(\varphi_k)$$

où φ est la phase.

Pour reconstruire un nombre complexe : x+y*1j ou module * np.exp(1j*phase)

(Si vous n'avez calculé que la moitié du spectre, il faut rétablir l'autre moitié à cet endroit)

Rappel : il faut utiliser le spectre d'amplitude sans le log

8. Estimation du spectre d'amplitude du bruit :

a. Cela consiste à réaliser la moyenne sur les 4-5 premiers spectres (coefficients par coefficients) et retourne un spectre de d'amplitude moyenne (du bruit).
b. Pour réaliser le calcul :
   - soit vous restez dans la boucle principale (juste après le calcul du spectre d'amplitude) et vous calculez à la volée la moyenne au fur et à mesure (sur les 5 premiers spectres),
   - soit vous faites ce calcul indépendamment (ce qui conduit à faire une boucle de traitement à l'extérieure de la boucle principale : fenêtrage, calcul Fourier et calcul du spectre d'amplitude).

Ce principe d'estimation du bruit suppose bien sûr que la parole ne commence pas pendant les premiers instants du signal (seul le bruit est présent).
[Prenez les fichiers *_bruit_ndB.wav]

9. Débruitage par soustraction spectrale

Réalisez une fonction de calcul de **soustraction spectrale** des spectres d'amplitude : elle soustrait au spectre d'amplitude le spectre de l'estimation du bruit.
La formule généralisée de la soustraction spectrale est la suivante :

$$\hat{Y}_k = \left(\hat{X}_k^{\alpha} - \beta \hat{B}_k^{\alpha}\right)^{1/\alpha} \quad si \ \text{la différence} > 0$$

$$\hat{Y}_k = \gamma \hat{B}_k \qquad \text{sinon}$$

avec X_k le spectre d'amplitude à l'instant k, B_k le spectre d'amplitude du bruit estimé (à l'instant k, mais qui est ici le même pour tous les k) et Y_k est le spectre d'amplitude résultat (à l'instant k).
Dans un premier temps, nous prendrons puissances α=2 et β=1, et le coefficient γ=0.

Une fois le spectre d'amplitude « nettoyé », en le recombinant avec le spectre de phase, vous obtenez le signal débruité.

Écoutez le résultat et jouez un peu sur les paramètres. β représente la force de la soustraction et le coefficient γ est présent (>0) pour éviter des portions de fréquences totalement vides en énergie (ces portions créent des îlots d'énergie de fréquence qui entrainent un bruit dît musical en haute fréquence).

RENDU pour le vendredi 14 avril (inclus)

Réalisez un mini-rapport :

[L'idée générale du rapport est de suivre une démarche scientifique : J'ai un algo à tester pour résoudre un problème, je présente l'algo, je mets en place des tests et j'évalue les résultats et je conclus sur l'algo]

- Réalisez un **diagramme/schéma** qui présente les différentes étapes de la soustraction spectrale (on part d'un signal, on le fenêtre, puis….)
- Présentez **différents tests** avec une brève analyse des résultats obtenus pour chacun suivant différents points de vue. Exemple : qualité du signal-résultat et qualité du débruitage (c-a-d est-ce que la parole est bien débruitée et aussi toujours intelligible…). Vous pourrez conclure de manière globale (y–a-t-il une combinaison optimale, ou un compromis, ou certains cas de bruit devrait mieux fonctionner…) : Faites des tests pour 2-3 valeurs de α (ex : 1, 2 et 4 par exemple), pour 2-3 valeurs de β (ex : 1,10,30) et pour 2-3 valeurs de γ (ex : 0,1, 3), sur les différents fichiers bruités.
- Il est possible bien sûr d'y inclure les spectrogrammes des fichiers débruités (avec la fonction savefig, ou sur l'interface de sortie des figures générées).