TP2 : Débruitage de la parole

(Fourier, Soustraction spectrale)

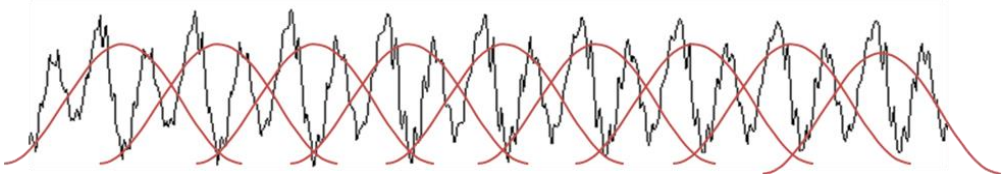
[le travail à rendre contiendra le fichier .java, .class et l'analyse du débruitage (.odt, .doc ou .pdf), le tout compressé dans un fichier .zip]

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 : langage JAVA, extrait de code pour l'ouverture/écriture d'un fichier .wav et classe FFT.class. 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), vous utiliserez le fichier test_seg.wav. 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. Compilez la classe SoundSignal et manipulez-là.
   [compilation : javac SoundSignal.java
   Execution : java SoundSignal test_seg.wav]



2. 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.
   `Double[] fenetrageHamming(int size)` Avec `size` la taille de la fenêtre
   Exemple pour calculer une fenêtre de Hamming :

```
        double c1 = (double) 0.54 ;
        double c2 = (double) 0.46 ;
        hamming = new double[size] ;
        for (int i = 0 ; i < size ; i++){
              hamming[i] = c1 - (double) (c2 * Math.cos(
(double) 2 * Math.PI * i / (size-1))) ;
        }
```

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, en parallèle, construisez un nouveau tableau de même taille (ex : `somme_hamming`) en y ajoutant des fenêtres de Hamming. Ici, le recouvrement est constant (sauf aux extrémités), don la valeur du tableau doit être constante (sauf aux extrémités) [ici 2.16].
Puis au final 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.

Votre programme aura donc la structure suivante :

Pour i=0 à i<taille_signal-n+1 ; i+=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 (5.)
- Traitement sur le spectre d'amplitude (7. 8.)
- Reconstruction du spectre (6.)
- Fourier inverse (3.)
- ***Reconstruction du signal (2.)***

Fin pour

Les étapes 2., et 3. sont des opérations « symétriques », elles s'opèrent en début et en fin de boucle.

3. Sur chaque fenêtre de signal, nous allons maintenant calculer la **transformée de Fourier**. Nous utiliserons pour cela, la classe FFT fournie (format .class ou .jar).

```
     import fft.FFT; //si .jar

     FFT fft = new FFT(fftOrder); //fftOrder : ordre de fft
                                  //    une puissance de 2
     double[] x = new double [fftOrder*2];

     //Initialisation du tableau x
     for (int i = 0 ; i < fftOrder ; i++){
           x[i*2] = signal_fen[i]; //signal fenétré
(attention à la taille de signal_fen)
           x[i*2+1] = 0;
           }


     //Réalisation de la transformée de Fourier
     fft.transform(x,false);
     //la partie réelle :     x[i*2]
     //la partie imaginaire : x[i*2+1]

     // (*)

     //Réalisation de la transformée de Fourier Inverse
     fft.transform(x,true);
     //le signal fenêtré final se trouve en x[i*2]
```

Nous voulons réaliser une FFT sur 1024 points, il faut donc x[1024*2].
Comme le signal est une fenêtre de 32 ms (~705 échantillons), il faut compléter par des zéros pour atteindre 1024 échantillons.
Réalisez la transformée d'une fenêtre et ensuite sa transformée inverse. On doit obtenir le même signal.

(*) les calculs sur le spectre se feront à cet endroit lors des étapes suivantes.

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

double [] spectreamplitude(double[] x, int fftorder) ;

Il faut calculer le module du spectre :

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

La fonction prendra en entrée le tableau du spectre de Fourier.
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 :

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

Mais nous ne ferons pas ce calcul ici.

5. 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** :
   double [] spectrephase(double[] x_fourier, int fftorder) ;
   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)
   Nous utiliserons la fonction Math.atan2(im, re) qui calcule directement la phase d'un complexe (re+i im).
   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 de phase est asymétrique).

6. **Reconstruire un spectre complexe** à partir du module et de la phase :
   double [] spectrereconstruction (double[] spectre_amplitude, double[] spectre_phase, int fftorder) ;
   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.

   Le tableau retourné a pour taille 2**fftorder* (*x*[2i] pour la partie réelle et *x*[2i+1] pour la partie imaginaire) comme pour 4. .
   (Si vous n'avez calculé que la moitié du spectre, il faut rétablir l'autre moitié à cet endroit)

   Rappel : Si vous aviez calculé le spectre d'amplitude avec un log, il aurait fallu revenir au module : $|X_k(\omega)| = \exp(spectre\_amplitude[k]/20)$

7. **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]

8. 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 :

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

$\hat{Y}_k = \gamma \hat{B}_k$ 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).

Réalisez un rapport :

- 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.