TP 6 – Traitement de sons réels

S.Gibet

Année 2022-2023

L'objectif est ici de bien comprendre les bases de la transformée de Fourier appliquée à des signaux réels. Le début du programme en Python *fft-sons* est fourni sur Moodle. L'idée est de le compléter afin d'effectuer des traitements sur les sons choisis.
Des données de sons vous sont données sur Moodle. À la fin du TP vous devrez avoir terminé la partie I du TP. Vous pouvez utiliser vos propres données sonores enregistrées en utilisant par exemple le logiciel *Audacity* (Partie II)

Venez en TP avec un casque.

Les deux parties qui suivent sont à terminer à la maison. Documentations : `https://sourceforge.net/projects/audacity/`
`https://docs.scipy.org/doc/scipy/reference/signal.html`

Partie I – Analyse de signaux sonores réels

Cette partie est à rendre à la fin du TP. Vous prendrez dans cet exercice des signaux sonores de votre choix de l'archive `Sons`. Vous utiliserez les librairies `numpy`, `matplotlib` et `scipy` pour lire un fichier son, traiter le signal et tracer le spectrogramme : voir le fichier `fft-sons.py`.

1. Lire un signal (chant d'oiseau par exemple). Affichez sa taille (en nombre d'échantillons), sa fréquence d'échantillonnage, ainsi que la durée du signal en secondes.

2. Tracez le spectrogramme du signal (encore appelé sonagramme pour le son). Vous pourrez utiliser les lignes de code suivantes :

```
f, t, Sxx = signal.spectrogram(x, fs)
plt.pcolormesh(t, f, np.log(Sxx))
```

3. Ecrire une fonction qui permet d'effectuer la transformée de Fourier de votre signal pour une fenêtre de *Nf* échantillons (prendre 512 ou 1024) et un offset temporel que vous choisirez (cela dépend du signal que vous utilisez).

4. Appelez cette fonction et tracez le spectre en amplitude pour différentes fenêtres temporelles. Qu'observez-vous ?

Partie II – Audacity

On vous propose dans cette partie d'enregistrer votre voix à l'aide du logiciel Audacity. Audacity vous permet d'apporter des modifications (égalisation, amplification, débruitage,...) qui améliorent la qualité des signaux enregistrés et permettent de faire du montage audio et vidéo.

Cette partie est à rendre à la fin de la semaine. Vous pourrez la préparer en téléchargeant sur votre ordinateur le logiciel Audacity et en enregistrant une ou plusieurs séquences sonores.

1. Récupérez une portion du signal enregistré sur 5 secondes.
2. Transformez et enregistrez quatre versions du même signal :
   - une version bruitée, en rajoutant du bruit,
   - une version où vous appliquez un filtre passe-bas qui laisse passer les basses fréquences et coupe les hautes fréquences,
   - une version où vous appliquez un filtre passe-haut qui laisse passer les hautes fréquences et coupe les basses fréquences,
   - une version améliorée : signal amplifié + égalisé + débruité.
3. Tracez les signaux audio à l'aide d'Audacity.
4. Puis, à l'aide des fonctions Python écrites dans la partie I, tracez l'un sous l'autre les 4 spectrogrammes des 4 signaux. Pour cela, vous utiliserez `subplot` (librairie matplotlib).

Partie III – Fenêtre glissante (pour aller plus loin)

On souhaite calculer le spectre du signal précédent à certaines zones spécifiques d'intérêt (zones du spectrogramme où le son est présent). On se propose alors d'extraire un morceau du signal correspondant à un fichier son donné. Ce morceau est défini par un Offset (décalage temporel à partir du début du fichier son) et un nombre d'échantillons dans le temps (winSize). On souhaite également appliquer une fenêtre temporelle qui permet d'éviter les bords non linéaires de la fenêtre rectangulaire. Dans ce cas, on multiplie le signal par cette fenêtre temporelle "arrondie" (winType).

1. Écrivez une fonction *calculerSpectre(sig, ...)* qui retourne le vecteur fréquentiel et le vecteur du spectre calculé sur le signal *sig*. Vous pourrez visualiser l'amplitude du spectre en dB.
2. Écrivez une fonction Python *movingFFT* qui appelle la fonction *calculerSpectre* pour plusieurs morceaux successifs du signal, avec éventuellement une intersection non nulle entre les morceaux. Cette fonction prend pour paramètres :
   fileName : Le nom du fichier signal à charger
   winSize : la taille de la fenêtre de signal à traiter
   offset : la position de la fenêtre de signal à traiter
   winType : le type de la fenêtre à utiliser (rectangulaire, hamming, hanning)
3. Testez cette fonction pour les signaux de votre choix, avec différents offsets et différents types de fenêtres temporelles.