TP - Scalogrammes

Le code ainsi que les réponses aux questions 2.f), 3.e), 3.f) et 4. sont à envoyer à l'adresse `waldspur@di.ens.fr` le vendredi 17 avril au plus tard.

Vous trouverez les fichiers dont vous avez besoin pour le TP à l'adresse `http://www.di.ens.fr/~waldspur/tds/14_15_s2/tp2.html`.

Pour me faciliter la correction, pensez à :

- ne pas omettre les instructions d'affichage que l'énoncé vous demande de faire figurer dans votre code
- respecter les noms de fonctions et les formats d'entrée/sortie demandés
- m'envoyer les réponses aux questions dans un format lisible (texte ou pdf)

1. Fonctions auxiliaires

a) Définir une fonction `smooth` qui prend en entrée trois paramètres : un signal réel `f` (sous la forme d'un vecteur colonne), sa fréquence d'échantillonnage `freq` et une durée `T` (exprimée en secondes).

b) Dans la fonction `smooth`, calculer le signal `filter` tel que :

$$\forall k \in \left\{ \left[-\frac{N}{2} \right] + 1, ..., \left[\frac{N}{2} \right] \right\} \qquad \texttt{filter}[k] = \exp\left(-\left(\frac{k}{\texttt{T.freq}} \right)^2 \right)$$

où N est le nombre d'éléments de `f`, [.] désigne la partie entière et les indices sont comptés modulo N.

c) Diviser `filter` par la somme de ses éléments et renvoyer la convolution de `f` et `filter` (projetée sur l'ensemble des réels pour éviter les erreurs numériques).

Cette opération représente un moyennage local, sur un intervalle de temps caractéristique de l'ordre de 2`T`.

d) Définir une fonction `preprocess`, qui prend en entrée un signal `f` et sa fréquence d'échantillonnage `freq`, et renvoie un signal `f_out` :

$$\texttt{f\_out}[k] = \frac{\texttt{f}[k]}{\sqrt{\texttt{p}[k]}}$$

où `p` est la moyenne de $\texttt{f}^2$, calculée (à l'aide de la fonction `smooth`) sur un intervalle de temps de l'ordre de 2 secondes.

Cette fonction effectue une normalisation locale des signaux audio. Elle permet d'éviter aux algorithmes qui vont être implémentés dans la suite du sujet d'être trop sensibles aux variations d'intensité sonore (un instrument qui joue d'abord doucement puis plus fort, par exemple).

2. Transformée en ondelettes

a) Définir une fonction `wavelet_transform`, qui prend en entrée trois paramètres : un signal réel `f`, sa fréquence d'échantillonnage `freq` et une fréquence `freq_out` (qui représente la fréquence à laquelle on échantillonnera la transformée en ondelettes).
b) Calculer la transformée de Fourier discrète de `f`. Déterminer l'indice `ind_max` de cette transformée de Fourier qui correspond à la fréquence `freq_out` et tronquer la transformée de Fourier pour ne garder que les indices inférieurs ou égaux à `ind_max`.
c) Calculer le vecteur suivant :

$$\forall j \in \{1, ..., 61\} \quad \texttt{f0}[j] = 440.2^{(j-25)/8}$$

Ce vecteur représente les fréquences caractéristiques des 61 ondelettes qui vont être utilisées pour calculer la transformée en ondelettes. La fréquence la plus basse est 55 Hz et la plus haute est autour de 10000 Hz.
d) Calculer un tableau `psi` de taille `ind_max`$\times 61$, tel que :

$$\texttt{psi}[k, j] = \frac{e^{30i\frac{\nu[k]}{\texttt{f0}[j]}}}{\left(1 + 6i\left(\frac{\nu[k]}{\texttt{f0}[j]} - 1\right)\right)^6}$$

où, pour tout k, $\nu[k]$ désigne la fréquence auditive correspondant à l'indice k de la transformée de Fourier discrète de `f`.
e) Renvoyer la transformée en ondelettes de `f`, sous la forme d'un tableau de taille `ind_max`$\times 61$ dont la j-ième colonne est la convolution de `f` avec le filtre ayant `psi`$[:, j]$ pour transformée de Fourier.

> Pour des signaux d'une durée d'environ 10 secondes, le calcul peut prendre quelques secondes (mais pas davantage).

f) Afficher le module de la transformée en ondelettes du signal `chirp.wav`, avec une fréquence d'échantillonnage égale à 10000 Hz. Comparer le résultat avec le spectrogramme obtenu dans le TD 6 pour le même signal ; décrire et expliquer la ou les principales différences.

> Pour afficher la transformée en ondelettes, on aura intérêt à transposer le tableau renvoyé par `wavelet_transform` (pour que l'axe des abscisses représente le temps et l'axe des ordonnées la fréquence) puis à lui appliquer `flipud`, de façon à ce que les hautes fréquences soient affichées en haut et les basses fréquences en bas.

3. Détection de notes

Dans cette question, on définit une fonction qui détecte les notes dans un morceau de musique.
a) Définir une fonction `detect_onsets`, qui prend en entrée un signal `f` et sa fréquence d'échantillonnage `freq`.
b) Appliquer à `f` la fonction `preprocess`.
c) Calculer `S`, la transformée en ondelettes de `f`. On pourra utiliser une fréquence d'échantillonnage égale à 10000 Hz.

d) Calculer `f_dec` :

$$\texttt{f\_dec}[k] = \sum_{j=1}^{61} |\texttt{S}[k, j]|$$

Afficher `f_dec` sur une nouvelle figure.

Utiliser `figure ;` pour ouvrir une figure vide.

Le principe de l'algorithme de détection est que, lorsqu'une note est jouée, la puissance sonore du signal doit augmenter, et ce dans plusieurs bandes de fréquences simultanément. Les débuts de notes sont donc à peu près les endroits où la fonction `f_dec` présente des pics. Une fois que `f_dec` a été calculée, la suite de l'algorithme consiste simplement à détecter correctement ses pics.

e) Remplacer `f_dec` par sa moyenne locale (calculée sur un intervalle de temps de l'ordre de 2×50 ms). Quelle est l'utilité de cette opération ?

f) Détecter les indices k auxquels `f_dec` atteint un maximum local et qui vérifient de plus :

$$\texttt{f\_dec}[k] > \texttt{mean\_f}[k] + 0,2$$

où `mean_f` est la moyenne locale de `f_dec` (à nouveau calculée sur un intervalle de temps de l'ordre de 2×50 ms).

Quelle est l'utilité de cette dernière condition ?

> La fonction `find`, appliquée à un tableau, renvoie les indices de toutes les cases du tableau contenant une valeur non-nulle.

g) Renvoyer les valeurs temporelles (exprimées en secondes) correspondant à ces indices.

h) À l'aide de la fonction `compare` fournie, afficher les performances de l'algorithme pour les signaux `rmc026.wav` et `rmg008.wav`.

> La fonction `compare` a besoin de deux arguments. Le premier est la liste des notes renvoyée par la fonction `detect_onsets`. Le deuxième est le nom du fichier dans lequel se trouvent les positions exactes ('rmc026.wav' pour le premier signal et 'rmg008.wav' pour le deuxième signal). Elle affiche deux nombres : le premier est la précision de la détection (c'est-à-dire la proportion de détections correspondant réellement à des notes) et le deuxième est le recall (c'est-à-dire le pourcentages de notes qui ont bien été détectées). À titre indicatif, ma propre implémentation donne 97.3% de précision et 100% de recall pour le premier signal, 100% de précision et 72.9% de recall pour le deuxième.

4. Classification

Écrire une fonction `classify` qui prend en entrée le nom d'un fichier audio (au format `wav`) et renvoie une chaîne de caractères égale à "piano" ou à "violon" selon que le signal audio représente du piano ou du violon. (On tolèrera que le violon soit remplacé ou accompagné par un alto ou un violoncelle.)

Quelques exemples pour vous entraîner sont disponibles à l'adresse donnée en début d'énoncé. Il n'est pas nécessaire que votre fonction les classifie tous correctement ; il faut simplement qu'elle fasse mieux qu'une classification aléatoire.

Décrire brièvement le principe de la méthode employée. Expliquer éventuellement pour quels types de signaux elle fonctionne moins bien.

[Indication : Une possibilité est d'utiliser le fait que les débuts de notes sont plus brusques dans le cas du piano que dans celui du violon. Cela peut se détecter à partir de la transformée en ondelettes.]