TP10 – Compression audio

Transformée de Fourier à court terme d'un signal acoustique

La première *représentation temps-fréquence* d'un signal temporel a été proposée en 1946 par Dennis Gabor, physicien hongrois ayant obtenu le prix Nobel en 1971 pour l'invention de l'holographie. La *transformation de Gabor* consiste à multiplier le signal par une fenêtre glissante afin d'obtenir le « spectre instantané » du signal. Bien que Gabor ait utilisé une gaussienne comme fenêtre glissante, il est possible d'utiliser n'importe quelle fenêtre. Cette généralisation s'appelle la *transformation de Fourier à court terme* (TFCT, *Short Time Fourier Transform* en anglais). La transformée de Fourier à court terme Y d'un signal unidimensionnel *continu* y s'écrit :

$$Y(\tau, f) \coloneqq \mathbf{TFCT}\,\{y\}\,(\tau, f) = \mathbf{TF}\{y\, w_\tau\}(f)$$

où w_τ désigne la fenêtre glissante, qui peut être positionnée à un instant τ variable. Par conséquent, alors que le signal d'origine y dépend d'une seule variable réelle (le temps t), sa transformée de Fourier à court terme Y, qui est une fonction complexe, dépend de deux variables réelles (la position temporelle τ de la fenêtre glissante et la fréquence f), obtenue par concaténation de spectres instantanés.

Dans le cas de signaux discrets $\mathbf{y} \in \mathbb{C}^L$, en choisissant une fenêtre $\mathbf{w} \in \mathbb{R}^N$ et un décalage H entre deux positions successives de la fenêtre, nous obtenons :

$$\mathbf{Y}(m, k) \coloneqq \mathbf{TFCT}\,\{\mathbf{y}\}\,(m, k) = \mathbf{TFD}\{\mathbf{y}\, \mathbf{w}_{mH}\}(k)$$

où $k \in \{0, \ldots, N-1\}$ et $m \in \{0, \ldots, \lfloor \frac{L-N}{H} \rfloor + 1\}$.

Comme les signaux manipulés sont réels, nous pouvons conserver uniquement les coefficients de Fourier correspondant aux fréquences positives ($k \in \{0, \ldots, N/2\}$), auquel cas le nombre de lignes de $\mathbf{Y}$ est égal à $N/2 + 1$. Quant au nombre de colonnes, il est égal au nombre de positions de la fenêtre glissante, qui dépend de la valeur H du décalage.

La représentation standard d'une transformée de Fourier consiste à afficher le *spectre de puissance* en utilisant une échelle en *décibels* (dB) :

$$S(\tau, f) \coloneqq |Y(\tau, f)|^2 \quad \Longrightarrow \quad S_{dB}(\tau, f) \coloneqq 10 \times \log_{10} |Y(\tau, f)|^2$$

Une telle représentation est appelée *spectrogramme* de manière générale, et *sonagramme* dans le cas de données audionumériques (cf. l'exemple de la figure 1).

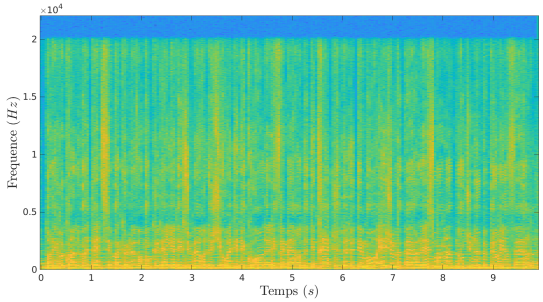

FIGURE 1 – Sonagramme d'un extrait audio (représentation en fausses couleurs).

Exercice 1 : calcul de la transformée de Fourier à court terme

Commencez par lancer le script `lecture`, qui lit un enregistrement sonore au format `WAV`. Par la suite, vous pourrez bien sûr tester d'autres fichiers audio, par exemple ceux du répertoire `Audio`.

Écrivez la fonction `TFCT`, appelée par le script `exercice_1`, permettant de calculer et d'afficher la transformée de Fourier à court terme du signal passé en paramètre. Notez l'influence des différents paramètres du script `exercice_1`, en particulier le type de fenêtre utilisée (fenêtre rectangulaire ou fenêtre de Hann).

Remarques importantes :

- Il est conseillé de découper le signal à l'aide de la fonction `buffer` de Matlab.
- Les colonnes de la matrice `TFCT` étant calculées à l'aide de la fonction `fft` de Matlab, les lignes de cette matrice correspondent aux fréquences, mais sont rangées dans l'ordre suivant : $f = 0, \ldots, f_{\max}$ pour la première moitié, $f = -f_{\max}, \ldots, 0$ pour la deuxième moitié. Il est demandé de ne garder que la partie correspondant aux fréquences positives (les autres informations étant redondantes si le signal est réel).
- Par défaut (ou en spécifiant `axis ij`), les axes des graphiques affichés par Matlab sont orientés vers la droite pour les abscisses, mais vers le bas pour les ordonnées. La commande `axis xy` permet de forcer l'orientation de l'axe des ordonnées vers le haut.
- Dans la mesure où la fonction `fft` de Matlab, appliquée à une matrice, calcule la TFD colonne par colonne, il est conseillé d'écrire la fonction `TFCT` sans boucle `for`.

Dans le script `exercice_1`, les décalages successifs de la fenêtre utilisée pour le calcul de la transformée de Fourier à court terme sont choisis de manière à former une partition du signal. La transformation de Fourier étant inversible, cette transformée de Fourier à court terme doit permettre de restituer le signal acoustique d'origine sans aucune distorsion. En revanche, cela n'est plus le cas si l'on dispose uniquement de la partie réelle de la transformée de Fourier à court terme (perte de la partie imaginaire) ou de son module complexe (perte de la phase). Le script `restitution_1` vous permet de comparer le son restitué dans ces trois cas de figure.

Exercice 2 : compression acoustique

Il est envisageable de tirer parti des limitations de nos capacités auditives pour compresser des données audio.

Tout d'abord, il est possible de tronquer le sonagramme pour ne garder que les fréqeunces inférieures à un certain seuil. En effet, notre oreille perçoit moins bien les sons aigus, et l'« adulte moyen » n'entend plus les sons au-delà de $16kHz$. De plus, un sonagramme nous indique quelles fréquences contribuent le plus à la reconstruction du signal à un instant donné. Il est donc possible de conserver uniquement une faible proportion des plus grands coefficients de Fourier, en espérant que la restitution du signal soit « satisfaisante ».

Le script `exercice_2` est censé détecter, dans chaque colonne d'un sonagramme tronqué, les `m` coefficients de Fourier les plus élevés (en module). Écrivez la fonction `mp3`, appelée par ce script, qui retourne deux matrices `indices_max` et `valeurs_max` comportant `m` lignes chacune : pour chaque colonne, `indices_max` contient les numéros de lignes des `m` plus grands coefficients de Fourier (en module), et `valeurs_max` contient les valeurs de ces coefficients.

Le script `restitution_2` permet d'évaluer l'impact de cette perte d'information sur le signal sonore restitué. En testant différentes valeurs des paramètres `seuil` et `m`, vous constaterez que le taux de compression du signal sonore peut atteindre une valeur avoisinant $1/10$, tout en garantissant une bonne restitution sonore. Vous venez de réaliser un système de compression acoustique simplifié (auquel il manque le codage des coefficients de Fourier sélectionnés).

Exercice 3 : stéganographie acoustique

La *stéganographie*, du grec *steganos* (« couvrir ») et *graphō* (« j'écris »), désigne l'acte de dissimuler un message dans un autre. Elle ne doit pas être confondue avec la cryptographie. Dans ce dernier cas, on n'ignore pas la présence du message, mais on n'a pas la clé pour le décrypter. La stéganographie permet de faire passer un message secret, ou de marquer des données de manière invisible, afin de pouvoir en repérer les copies illégales.

La stéganographie acoustique consiste donc à cacher un enregistrement sonore dans un autre. Plutôt que de compresser les données en éliminant une certaine proportion des hautes fréquences, la place libérée peut être mise à profit pour stocker les coefficients de Fourier des basses fréquences d'un autre enregistrement sonore.

Lisez le fichier audio `message_cache.wav` à l'aide du script `lecture`. Ce fichier recèle un enregistrement sonore caché. À vous de le trouver en écrivant un script dédié, de nom `exercice_3`. Indication : pour rendre l'enregistrement sonore caché indétectable, ses coefficients de Fourier ont été divisés par 1000.

Exercice 4 : débruitage audio (facultatif)

Le logiciel libre *Audacity* comporte une fonctionnalité permettant d'atténuer le bruit d'un signal sonore. Il s'agit d'une méthode appelée *spectral gating*, où un modèle fréquentiel de bruit permet de garder ou d'atténuer les coefficients du sonagramme du signal bruité.

Supposons que nous disposions d'un extrait de signal comportant uniquement du bruit (extrait qui nous permet de créer un « modèle » de bruit). L'algorithme utilisé procède comme suit :

1. Calcul du sonagramme de l'extrait de bruit.
2. Calcul de la moyenne μ et de l'écart-type σ temporels des coefficients de ce sonagramme (μ et σ sont deux vecteurs de même taille, égale au nombre de lignes du sonagramme).
3. Choix d'un seuil déterminé grâce à ces valeurs. Il est conseillé de choisir le seuil égal à $\mu + \alpha\,\sigma$, où α est un coefficient permettant de régler la sensibilité du filtrage (le seuil lui aussi est un vecteur).
4. Comparaison de chaque coefficient du sonagramme au seuil : le masque ainsi obtenu vaut 0 si le coefficient est inférieur au seuil, 1 sinon. À ce niveau de l'algorithme, il est possible d'utiliser la fonction `imgaussfilt` de Matlab pour « lisser » le masque, c'est-à-dire pour rendre le filtrage moins binaire.
5. Enfin, application de ce masque à la TFCT du signal bruité.

Le script `exercice_4` génère un bruit qui est ajouté au signal lu par le script `lecture`. Il appelle ensuite la fonction `debruitage`, à laquelle sont passées en paramètres la TFCT du bruit et la TFCT du signal bruité. Écrivez cette fonction, qui est censée retourner la TFCT du signal débruité.

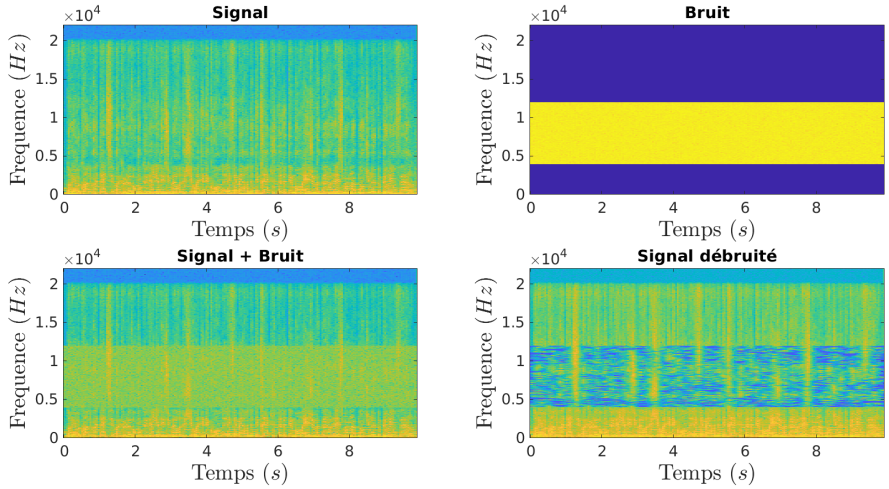

FIGURE 2 – Débruitage d'un signal audio (artificiellement) bruité.