TP11 – Transformation de Gabor

La première représentation temps-fréquence d'un signal temporel est due à Gabor, physicien hongrois ayant obtenu le prix Nobel en 1971 pour l'invention de l'holographie. La *transformation de Gabor*, qu'on appelle également *transformation de Fourier locale* ou *transformation de Fourier à fenêtre glissante*, est bien antérieure (1946) à la transformation en ondelettes (1984). Elle consiste à multiplier le signal par une *fenêtre glissante*, afin d'obtenir l'*analyse fréquentielle instantanée* d'un signal.

Bien que Gabor ait utilisé une gaussienne comme fenêtre glissante, on utilise plus souvent une « fonction créneau ». La transformée de Gabor $TG\{s(t)\}$ d'un signal temporel $s(t)$ s'écrit donc :

$$TG\{s(t)\}(\tau, f) = TF\{s(t)w(t-\tau)\}(f)$$

où TF désigne la transformée de Fourier et $w(t-\tau)$ la fenêtre glissante, qui peut être positionnée à un instant τ variable. Par conséquent, cette transformée est une fonction de deux variables (τ, f), où τ désigne le temps et f la fréquence. Sa représentation se fait donc généralement dans le plan (temps en abscisse et fréquence en ordonnée), sous la forme d'une pseudo-image dont le niveau de gris est proportionnel à la partie réelle. Une telle pseudo-image s'appelle un *sonagramme* (ou *spectrogramme sonore*). Les sonagrammes sont très commodes, par exemple, pour caractériser des voix ou des instruments de musique, dans la mesure où ils constituent des « empreintes acoustiques ».

Exercice 1 : transformée de Gabor d'un enregistrement musical

Commencez par lancer le script `musique.m`, qui lit un fichier audio au format `WAV` (vous pourrez ensuite tester vos scripts sur les autres fichiers `WAV` du répertoire `Audio`).

Complétez ensuite le script `exercice_1.m` afin de calculer et d'afficher la transformée de Gabor de cet enregistrement musical. Pour l'affichage, il est fait appel à la fonction `centrage.m` (cette fonction est fournie) car, par défaut, la fonction `fft` de Matlab ne situe pas la fréquence nulle au centre du spectre.

La transformation de Fourier étant inversible, si les différentes fenêtres utilisées pour le calcul de la transformation de Gabor sont disjointes et constituent une partition du signal, alors on peut l'inverser en utilisant la transformation de Fourier inverse (fonction `ifft` de Matlab). C'est ce que se propose de faire le script `exercice_1.m` en guise de vérification.

Dans la suite de ce TP, vous allez voir jusqu'à quel point il est possible de condenser une transformée de Gabor tout en continuant à pouvoir restituer la musique d'origine sans trop de dommages. Commencez par modifier le script `exercice_1.m` de manière à ne conserver que la partie réelle de la transformée de Gabor. Le signal restitué n'est-il pas trop dégradé ? Faites de même en ne conservant que le module complexe.

Exercice 2 : calcul du sonagramme à partir de la transformée de Gabor

Une autre façon de condenser la transformation de Gabor est envisageable : on peut supprimer les coefficients du spectre qui correspondent aux fréquences négatives et ceux qui correspondent aux hautes fréquences. Cela cause évidemment des dommages irréversibles au signal, mais n'en cause pas forcément au son perçu par l'oreille humaine. En revanche, c'est en procédant ainsi qu'on peut envisager de compresser le signal (nous reviendrons sur ce point dans l'exercice 4).

Complétez le script `exercice_2.m`, qui ne garde de la transformée de Gabor que les coefficients correspondant à une proportion (modulable) des fréquences positives. Jusqu'à quel pourcentage minimal de coefficients conservés le son restitué demeure-t-il fidèle à l'original ? Comme pour l'exercice 1, refaites ces tests en n'utilisant plus que la partie réelle des coefficients de la transformée de Gabor, voire son module complexe.

Exercice 3 : création de la partition à partir du sonagramme

Vous venez de voir qu'un ordinateur permet de « jouer » le sonagramme d'un morceau de musique de manière très fidèle à l'original, pourvu que le nombre de coefficients éliminés dans les hautes fréquences ne soit pas trop élevé. La *partition musicale* est une représentation temps-fréquence beaucoup plus connue et beaucoup plus condensée que le sonagramme (cf. figure 1). Cependant, le signal restitué à partir d'une partition dépend fortement de l'instrument utilisé ou de l'interprète, même si c'est un ordinateur qui joue (cf. exercice 4). C'est la raison pour laquelle on parle de « l'interprétation » d'une partition.

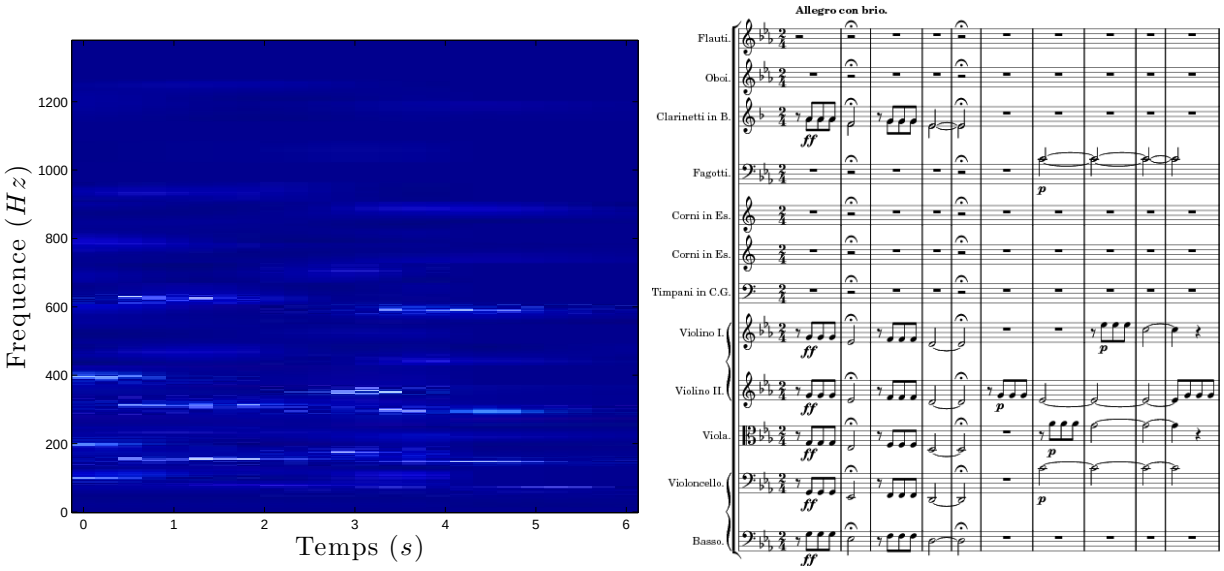

FIGURE 1 – Deux représentations temps-fréquence d'un même morceau de musique symphonique : sonagramme (à gauche) et partition (à droite).

Complétez le script `exercice_3.m` de façon à créer et à afficher la partition musicale correspondant au sonagramme réalisé précédemment. Vous devez déterminer, pour chaque mesure, la fréquence correspondant au module maximal du spectre. Les fonctions `frequences_ordonnees.m` (qui traduit les fréquences en positions verticales sur la portée) et `affichage_partition.m` (qui trace la portée, les notes et les silences) sont fournies.

Exercice 4 : application à la compression audio

Un « synthétiseur » est un appareil électronique qui joue un morceau de musique à partir d'une partition. Le script `synthese.m` vous est fourni. Il « joue » la partition transcrite par `exercice_3.m`. Ce script procède comme suit : dans un premier temps, le sonagramme est reconstitué (tant bien que mal) à partir de la partition ; ensuite, la transformée de Gabor est reconstituée à partir du sonagramme, puis inversée (cf. script `exercice_2.m`). Vous notez que le résultat est très décevant. Cela est principalement dû à la perte d'une grande partie de l'information fréquentielle, mais aussi à ce que la partition ne contient aucune information sur la *phase* du signal.

L'exercice 4 vise à améliorer la fidélité au signal original du son restitué (sans viser pour autant la « haute fidélité » !). En vous inspirant des scripts `exercice_3.m` et `synthese.m`, écrivez un script `exercice_4.m` qui détecte, pour chaque mesure, non plus seulement la fréquence correspondant au module maximal du spectre, mais les `n` fréquences correspondant aux `n` plus grandes valeurs du module du spectre (plus précisément, celles qui parmi ces valeurs sont supérieures à `seuil_audible`). Il est conseillé d'utiliser la fonction `sort` de Matlab.

Relancez le script `exercice_2.m` en fixant la valeur de `proportion` à 1, puis testez le script `exercice_4.m` pour différentes valeurs des paramètres `seuil_audible` et `n`, par exemple 0 et 100. Vous constatez qu'en procédant ainsi, le coefficient de compression du signal audio peut atteindre des valeurs relativement élevées sans que la dégradation du signal soit perceptible. Vous venez de réaliser un système de compression audio analogue à celui de la compression MP3 (en un peu moins sophistiqué).