



UE MU4RBI08-audio
Introduction au traitement du signal audio
Cahier de TP

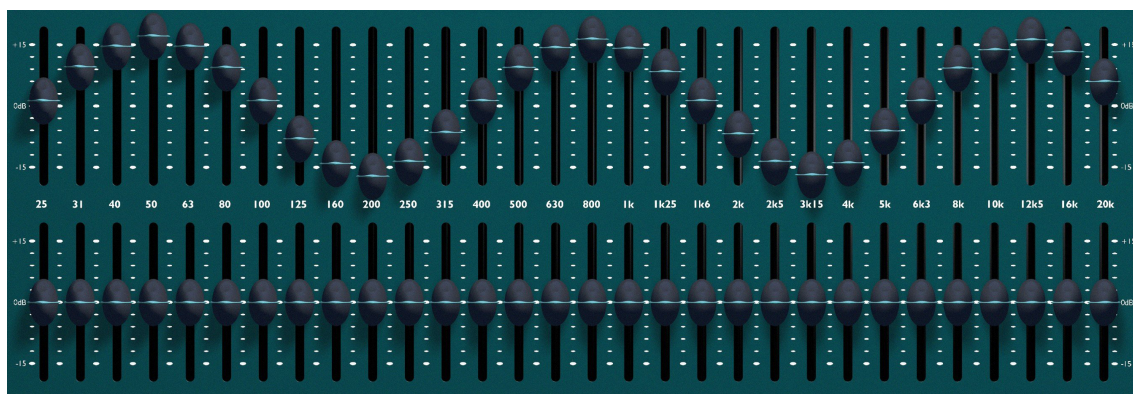
N. Obin, H. Boutin, S. Argentieri

Contexte

L'audio occupe aujourd'hui une place croissante dans les technologies numériques pour améliorer l'expérience de l'utilisateur avec les machines ou simuler des environnements numériques réalistes. Les smartphones et les assistants personnels sont ainsi équipés de systèmes de reconnaissance et de synthèse de la parole ; la réalité virtuelle exploite l'écoute binaurale pour spatialiser au casque des sources sonores ; les plateformes de musique en ligne utilisent des algorithmes de compression pour streamer la musique le plus efficacement possible, et utilisent des informations extraites du contenu musical pour identifier ou recommander un morceau.

Dans le cas des systèmes intelligents, l'information doit pouvoir être extraite d'un signal audio capté dans un environnement bruité et réverbéré. C'est notamment le cas des plateformes robotiques modernes destinées à évoluer dans des environnements inconnus et produisant leur propre bruit de fond, lié par exemple à leur mouvement (moteurs) ou leur refroidissement (ventilateurs).

L'extraction, l'analyse et le traitement de l'information d'un signal audio numérique dans de telles conditions fait appel à diverses connaissances et techniques qui seront abordées au cours de quatre séances de travaux pratiques.



Ces travaux pratiques ont pour objectif de fournir les bases du traitement de l'information audio, depuis leurs fondements théoriques jusqu'à leur mise en œuvre pratique. Ils porteront sur des exemples réalistes d'acquisition, d'analyse et de codage de signaux sonores. Les quatre séances seront organisées de la façon suivante :

- **TP 1** : Acquisition et quantification d'un signal audio. Rééchantillonnage et phénomène de repliement spectral.
- **TP 2** : Analyse temps-fréquence d'un signal audio. Transformée de Fourier à court-terme. Principe d'incertitude temps-fréquence. Précision et résolution fréquentielle.
- **TP 3** : Reconstruction du signal audio. Transformée de Fourier à court-terme inverse. Reconstruction par ajout et chevauchement.
- **TP 4** : Compression d'un signal audio. Codage perceptif. Allocation non uniforme de bits en fréquence.

Les séances de TP donnent lieu à une évaluation continue. Vous avez 4 rapports à remettre au total : un rapport le vendredi de la semaine où a lieu le TP (23h59). Chaque rapport sera limité à 12 pages et évalué sur les réponses aux questions : toute réponse devra être justifiée. Les 4 notes résultantes formeront votre note de TP. Les TPs seront implémentés de préférence dans le langage de programmation Python (avec notebooks pour les illustrations), ou en Matlab.

1 TP 1 : acquisition d'un signal audio

(Avant chaque écoute, veuillez à réduire le son du haut-parleur de votre ordinateur !)

1.1 Étude de la quantification

La fonction de quantification `quantif` vous est donnée. Elle réalise une quantification uniforme d'un signal x sur N bits pour une pleine échelle A . Les réponses reposeront sur des mesures expérimentales, réalisées à partir du fichier son donné. Elles seront comparées aux valeurs théoriques attendues.

1. Charger le signal audio x depuis le fichier `sine440.wav` : la fonction `wav.read('myfile')` de python importe les signaux '`wav`' codés au format 'PCM' compris entre -2^{N-1} et $+2^{N-1} - 1$, N étant le nombre de bits sur lequel sont codés les fichiers `.wav`, égal à 16. Pour être converti en 'float' entre -1 et $+1$, les signaux importés doivent donc être divisés par 2^{N-1} .
2. Pour $N = 4$ et $A = 1$, tracer le signal quantifié et le bruit de quantification.
3. Etudier les propriétés statistiques du bruit de quantification (moyenne, variance, etc.) et caractériser sa densité de probabilité (blancheur, etc.). Comparer les résultats obtenus aux hypothèses posées en cours.
4. Tracer l'évolution du rapport signal sur bruit de quantification RSB_q en fonction de N . Commenter la courbe expérimentale obtenue lorsque N croît ?
5. Pour $N = 8$ et pour différentes valeurs de A bien choisies, tracer le signal quantifié et le comparer à x .
6. (a) Fixer $N = 8$. Tracer le RSB en fonction de A , pour A variant sur un intervalle judicieusement choisi, puis en fonction du facteur de charge en dB : $10 \log(F^2)$, dépendant lui-même de A .
(b) Tracer d'autres courbes du RSB en fonction de $10 \log(F^2)$ pour d'autres valeurs de N . Interpréter l'allure de ces courbes.
(c) Quelles sont les conséquences d'une pleine échelle trop faible par rapport à la variance du signal x ?
7. Enfin, écouter l'effet de la quantification sur le signal audio `voice.wav` fourni en fonction de N . A partir de quelle valeur ne percevez-vous plus de différences avec le signal original ?

1.2 Rééchantillonnage et repliement spectral

Une rythmique a été enregistrée en studio, convertie en signal numérique et sauvegardée dans le fichier `rythmique.wav`. On souhaite étudier l'effet de la fréquence d'échantillonnage sur le contenu spectral du signal enregistré et du son restitué.

1.2.1 Rythmique

1. Charger le signal audio stéréo `rythmique.wav`, et préciser sa fréquence d'échantillonnage. Il sera noté x^R dans la suite du TP.
2. Afficher l'une des 2 voix du signal en fonction du temps.
3. Calculer et afficher son spectre : module en dB et phase, avec un axe fréquentiel linéaire. Sur quel intervalle de fréquence se trouve l'information (musicale) ? En déduire la fréquence d'échantillonnage minimale pouvant être utilisée pour un tel enregistrement. Elle sera notée F_s^{min} dans la suite du TP.
4. Afin de simuler un échantillonnage à la fréquence F_s^{min} , on choisit de sous-échantillonner le signal x^R d'un facteur N . Cette opération consiste à prélever dans x^R : 1 échantillon tous les N échantillons et à le copier dans un nouveau signal noté $x_{dec}^R : \forall k \in \mathbb{Z}, x_{dec}^R[k] =$

$$x^R[1 + (k - 1)N].$$

Déterminer le facteur N simulant un échantillonnage à la fréquence la plus proche possible de F_s^{min} . Construire le signal sous-échantillonné x_{dec}^R .

5. Comparer les signaux x^R et x_{dec}^R .
6. Calculer le spectre de x_{dec}^R (module et phase) sur un nombre de points fréquentiels égal à celui de x^R . Quel est l'effet du sous-échantillonnage sur l'amplitude et la position des composantes fréquentielles ?
7. La fréquence F_s^{min} permettra-t-elle, comme souhaité, de restituer le signal musical sans perte d'information ? Pour vérification, exporter x_{dec}^R en **.wav** à la fréquence de reconstruction F_s^{min} . Ecouter et comparer le signal audio obtenu au signal initial **rythmique.wav**.

1.2.2 Piano

1. Charger maintenant le signal audio **piano.wav** (ce signal est noté x^P) et répéter les étapes précédentes.
2. Quelle est la fréquence F_s^{min} minimale pouvant être utilisée pour échantillonner ce signal audio ?
3. Sous-échantillonner le signal à la fréquence F_s^{min} , puis à la fréquence 1024 Hz. Les signaux obtenus sont notés x_{dec}^{P1} et x_{dec}^{P2} . Comparer les spectres de x_{dec}^{P1} et x_{dec}^{P2} à celui de x^P .
4. Décrire le phénomène responsable des distorsions observées dans le spectre de x_{dec}^{P2} . Mesurer et justifier la position du pic de plus basse fréquence observé avec une échelle linéaire en ordonnée.
5. Exporter x_{dec}^{P1} et x_{dec}^{P2} en **.wav** aux fréquences de reconstruction correspondantes. Comparer les signaux audio au signal initial **piano.wav**.

1.2.3 BONUS

Reproduire la même démarche sur des enregistrements de violon **violon.wav**, d'orchestre symphonique **orchestre.wav** et de voix **voice.wav**. Visualiser et caractériser le contenu spectral des signaux audio étudiés.

2 TP 2 : Analyse temps-fréquence et principe d'incertitude

2.1 Transformée de Fourier à court-terme

La première partie sera dédiée à l'implémentation de la transformée de Fourier à Court-Terme (TFCT). La fonction, que l'on nommera `tfct.py`, aura pour paramètres d'entrée le chemin d'un fichier son, la taille de la fenêtre d'analyse N_{win} , le pas d'avancement de la fenêtre d'analyse N_{hop} , le nombre de points fréquentiels N_{fft} utilisés pour la TFD et la fréquence d'échantillonnage. On supposera dans toute la suite et par défaut que $N_{fft} = N_{win}$ et $N_{hop} = N_{win}/2$, et on travaillera avec une fenêtre de **Hamming**. Les paramètres de sortie seront la matrice TFCT du signal, notée x_{mat} , ainsi que les vecteurs fréquentiels et temporels utiles à l'affichage de la TFCT. Cette première partie vise à réimplémenter les différentes étapes de la TFCT.

1. Découpage du signal audio en trames de N_{win} échantillons avec un pas d'avancement de N_{hop} échantillons. Charger le signal audio `sound.wav` dans un vecteur qui sera noté x_{vect} dans la suite du TP. Isoler puis tracer l'allure temporelle d'une trame de x_{vect} , en fonction du temps en secondes. La comparer avec la même trame fenêtrée.
2. Transformée de Fourier Discrète (TFD) de la trame sur N_{fft} points fréquentiels. Tracer le spectre de la même trame de signal avant et après fenêtrage. On rappellera l'effet du fenêtrage sur la TFD.
3. Transformée de Fourier à Court Terme (TFCT). On se propose de calculer les TFD de chaque trame et de les ranger dans les L colonnes de x_{mat} .
Déterminer les dimensions de x_{mat} en fonction des paramètres d'entrée : le nombre de colonnes L correspond au nombre de trames à extraire dans x_{vect} et le nombre de lignes M correspond au nombre minimal de points fréquentiels utiles pour chaque spectre. Déclarer la matrice TFCT initialement nulle, puis remplir ses colonnes.
4. Test et affichage. Tester l'algorithme `tfct.py` sur le fichier audio `sound.wav`. Tracer le spectrogramme obtenu après calcul de la TFCT, à l'aide de la fonction python `plt.imshow`, en prenant soin de représenter l'axe des temps en secondes et l'axe des fréquences en Hertz.

2.2 Principe d'incertitude temps-fréquence

Nous allons maintenant étudier les propriétés temps-fréquence à partir de l'analyse de Fourier court-terme, donnant lieu au tracé d'un *spectrogramme*.

5. Après avoir chargé et écouté le fichier `diapason.wav`, analyser son contenu fréquentiel en dB avec la fonction `fft` de la librairie `numpy`. Que constatez-vous ? Quelle est la fréquence du son ? En déduire la note émise.
6. Même question avec le fichier `saxo.wav`. Que constatez-vous ? L'analyse précédente permet-elle de rendre compte de l'évolution du son (et de son contenu spectral) au cours du temps ? En quoi cela peut-il être pertinent ?

Dans la suite, nous appuierons notre analyse sur le spectrogramme du son étudié fourni par notre fonction `tfct.py` et le fichier son `piano.wav`.

7. *Précision fréquentielle* : fixer $N_{win} = 1024$. Faire varier N_{fft} , de sorte que :

- $N_{win} = 1024$,
- $N_{hop} = N_{win}$,
- $N_{fft} = X$,

avec $X \in [1024, 2048, \dots]$ la valeur numérique choisie. A l'aide du curseur, relever graphiquement la fréquence fondamentale du son en traçant le spectre d'amplitude **pour une seule trame judicieusement choisie** de la TFCT. A partir de quelle valeur de N_{fft} la mesure devient suffisamment précise ? Pourquoi ?

8. *Résolution fréquentielle* : augmenter N_{win} et N_{fft} , de sorte que :

- $N_{win} = X$,

$$— N_{hop} = N_{win},$$

$$— N_{fft} = X,$$

avec $X \in [128, 512, \dots]$ la valeur numérique choisie. A partir de quelle valeur peut on distinguer la fréquence fondamentale des harmoniques du son ? Expliquer en analysant l'impact de la fenêtre sur le spectre, et justifier par le calcul.

9. *Précision ET résolution fréquentielles* : fixer maintenant $N_{win} = 128$. Faire varier N_{fft} , de sorte que :

$$— N_{win} = 128,$$

$$— N_{hop} = N_{win},$$

$$— N_{fft} = X,$$

avec $X \in [128, 512, \dots]$ la valeur numérique choisie. Commenter l'effet de X sur l'allure du spectrogramme. N_{fft} permet-il d'améliorer la lecture de la fréquence fondamentale ? Est-il possible dans ce cas de mesurer la fréquence fondamentale du son ? Pourquoi ?

10. Conclure sur les conditions nécessaires et suffisantes pour mesurer la fréquence fondamentale du son.

11. Enfin, modifier les valeurs du paramètre N_{hop} , de sorte que :

$$— N_{win} = 1024,$$

$$— N_{hop} = X,$$

$$— N_{fft} = 2048,$$

avec $X \in [128, \dots, 1024]$ la valeur numérique choisie. Commenter l'effet de X sur l'allure du spectrogramme.

2.3 BONUS

En déterminant le bon jeu de paramètres $\{N_{win}, N_{hop}, N_{fft}\}$, utiliser le spectrogramme pour suivre la mélodie jouée au cours du temps du fichier son `saxo.wav`, en retrouvant les notes qui la compose (cf. Figure 1).

Note	Octave				
	1	2	3	4	5
<i>Do</i>	65,4064	130,813	261,626	523,251	1046,50
<i>Do#</i>	69,2957	138,591	277,183	554,365	1108,73
<i>Re</i>	73,4162	146,832	293,665	587,330	1174,66
<i>Re#</i>	77,7817	155,563	311,127	622,254	1244,51
<i>Mib</i>					
<i>Mi</i>	82,4069	164,814	329,628	659,255	1318,51
<i>Fa</i>	87,3071	174,614	349,228	698,456	1396,91
<i>Fa#</i>	92,4986	184,997	369,994	739,989	1479,98
<i>Sol</i>	97,9989	195,998	391,995	783,991	1567,98
<i>Sol#</i>					
<i>Lab</i>	103,026	207,652	415,305	830,609	1661,22
<i>La</i>	110,000	220,000	440,000	880,000	1760,00
<i>La#</i>					
<i>Sib</i>	116,541	233,082	466,164	932,328	1864,66
<i>Si</i>	123,471	246,949	493,883	987,767	1975,53

FIGURE 1 – Fréquences des différentes notes.

3 TP 3 : Reconstruction du signal audio : TFCT inverse

3.1 Implémentation de la TFCT inverse par Overlap-Add (OLA)

La première partie du TP consiste à implémenter l'algorithme de Transformée de Fourier à Court-Terme Inverse (ITFCT). La fonction, que l'on nommera `itfct.py`, aura pour paramètres d'entrée la TFCT d'un signal, le pas d'avancement de la fenêtre d'analyse N_{hop} , ainsi que le nombre de points fréquentiels utilisés dans l'algorithme TFCT N_{fft} et la fréquence d'échantillonnage F_s du signal initial. La TFCT, notée x_{mat} , sera écrite sous la forme d'une matrice de dimensions $n_l \times L$ déterminées au TP2. Les étapes de l'algorithme OLA sont résumées ci-dessous, et décrites dans la figure 2.

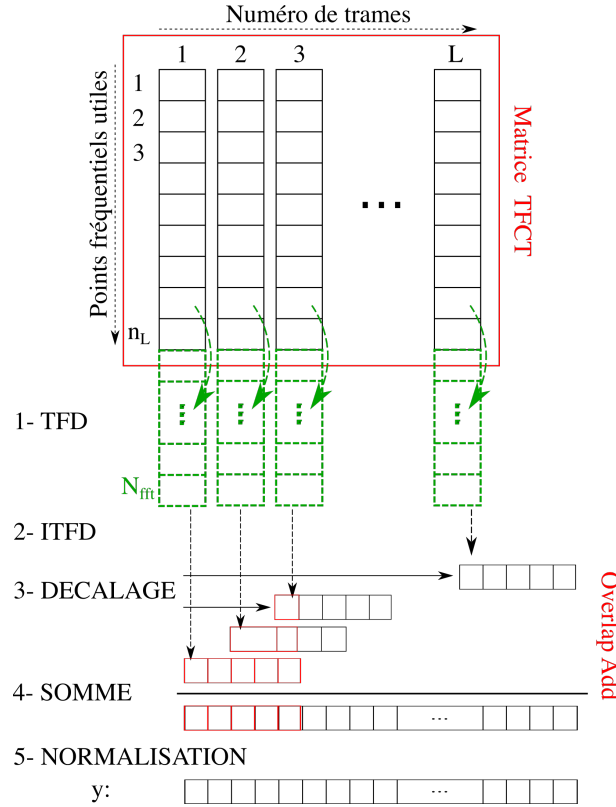


FIGURE 2 – Principe de l'algorithme OLA de la transformée de Fourier à court-terme inverse (ITFCT) pour la synthèse d'un signal sonore.

Etape 1 : reconstruire la TFD de chaque trame à partir des colonnes de la matrice x_{mat} de TFCT, dans des vecteurs de taille N_{fft} .

Etape 2 : reconstruire chaque trame fenêtrée en effectuant la TFD inverse (ITFD) des vecteurs de TFD : $\Rightarrow L$ trames fenêtrées $y_l, l \in [0, L - 1]$:

$$\begin{aligned} y_1[n] &= x[n]w[n] \\ y_2[n] &= x[n + N_{hop}]w[n] \\ &\dots \\ y_L[n] &= x[n + (L - 1)N_{hop}]w[n] \end{aligned}$$

La TFD inverse reconstruit des trames de longueur N_{fft} . Cette longueur est supposée égale à la longueur des trames découpées lors de l'algorithme TFCT (c'est le cas si la TFD a été effectuée sur un nombre correct de points).

Etape 3 : décaler la l^{e} trame y_l , de $(l - 1)$ trames \Rightarrow synchronisation des y_l , $l \in [1, L]$:

$$\begin{aligned} y_1[n] &\Rightarrow y_1[n] = x[n]w[n] \\ y_2[n] &\Rightarrow y_2[n - N_{hop}] = x[n]w[n - N_{hop}] \\ &\dots \\ y_L[n] &\Rightarrow y_L[n - (L - 1)N_{hop}] = x[n]w[n - (L - 1)N_{hop}] \end{aligned}$$

Etape 4 : sommer les trames décalées $y_l[n - (l - 1)N_{hop}]$

$$\sum_{l=1}^L y_l[n - (l - 1)N_{hop}] = x[n] \sum_{l=1}^L w[n - (l - 1)N_{hop}]$$

Etape 5 : normaliser le signal temporel. Les fenêtres usuelles sont choisies de sorte que la somme $K = \sum_{l=1}^L w[n - (l - 1)N_{hop}]$ soit indépendante de n . Dans ce cas (cf. Cours4bis) : $K \approx \sum_{n=0}^{N_{win}-1} w[n]/N_{hop}$ (égalité lorsque N_{win} est un multiple de N_{hop}). Alors

$$\frac{1}{K} \sum_{l=1}^L y_l[n - (l - 1)N_{hop}] \approx x[n]$$

Les paramètres de sortie de la fonction `itfct.py` seront le signal temporel reconstruit, noté y_{vect} et le vecteur temporel utile à son affichage. Cette première partie vise à réimplémenter les différentes étapes l'algorithme.

1. Préallocation de mémoire pour le signal reconstruit y : déterminer sa taille en fonction des paramètres d'entrée, puis déclarer le vecteur y initialement rempli de 0.
2. Reconstruction de chaque trame y_l à partir de sa TFD (colonne de x_{mat}).
3. Somme des trames reconstruites y_l à la bonne position de y .
4. Normalisation de y : division par $\sum_{n=0}^{N_{win}-1} w[n]/N_{hop}$. La fenêtre, correspondant à celle utilisée lors de l'algorithme `tftct`, est supposée être une fenêtre de Hamming de longueur $Nfft$.
5. Test de l'algorithme sur la matrice x_{mat} calculée par l'algorithme `tftct.py`, appliqué au fichier audio `sound.wav`. Préciser et discuter la valeur des paramètres utilisés pour la TFCT. Afin de comparer le résultat au fichier initial, calculer l'erreur quadratique entre ces deux signaux. Commenter.

3.2 Application au débruitage pour soustraction spectrale

La deuxième partie sera dédiée à la suppression de bruit stationnaire par soustraction spectrale. Les détails de l'algorithme sont décrits dans l'article :

Steven F. Boll (1979). *Suppression of Acoustic Noise in Speech Using Spectral Subtraction*. IEEE Transactions on Acoustics, Speech, and Signal Processing, vol.27(2), p.113-120.

7. Charger le signal audio bruité `mix.wav`. Calculer la TFCT X du signal bruité x à l'aide de l'algorithme implémenté au TP2, en choisissant les paramètres N_{win} et N_{hop} de manière pertinente.
8. Après avoir identifié une région de silence dans le spectrogramme du mélange $|X|$ (qui ne contient que le bruit de fond supposé stationnaire), créer le spectre d'amplitude du bruit $|x_{noise}|$. Tracer et commenter le spectre du bruit obtenu.

9. Estimer le spectrogramme du signal débruité $|X_{clean}|$ par soustraction de la signature spectrale du bruit au module de la TFCT du signal bruité :

$$|X_{clean}| = |X| - |X_{noise}|$$

On tracera et on comparera le spectrogramme avant et après soustraction du spectre de bruit.

10. On veillera particulièrement à effectuer le redressement des valeurs négatives à 0 après soustraction. Pourquoi est-ce important ?
11. Reconstruire le signal audio débruité, avec la fonction `tfct_inv` implémentée dans la partie précédente. Peut-on reconstruire le signal à partir de $|X_{clean}|$? Que constatez-vous ? Expliquer.
12. Reconstruire le signal audio débruité à partir du spectre complexe :

$$X_{clean} = |X_{clean}| e^{j \arg(X)}$$

Que constatez-vous ? Expliquer.

13. On jugera à l'écoute et on commentera la qualité du signal audio débruité. Le débruitage est-il parfait ? Pourquoi ? En bonus, on pourra mesurer et comparer les RSB entre le signal propre donné en accompagnement et les signaux avant et après suppression du bruit.

On illustrera et commentera chacune des étapes du traitement réalisé, et on joindra les signaux audio débruités réalisés au cours de cette partie.

4 TP 4 : Compression d'un Signal Audio

La compression de données est un domaine clef dans l'ère du numérique : pouvoir stocker, partager, et lire des données avec le minimum d'espace de stockage, le minimum de mémoire vive, et dans un minimum de temps. La compression audio avec perte (MPEG-1 layer I-II-III) repose sur une idée simple : on ne code pas ce qui ne s'entend pas. Ainsi, l'audio compressé est sensé être identique perceptivement à l'audio original. L'objectif de cette partie est d'étudier un algorithme de compression audio inspiré de la norme MPEG-1 layer III, avec un focus sur l'effet de l'allocation de bits sur la qualité de l'enregistrement audio.

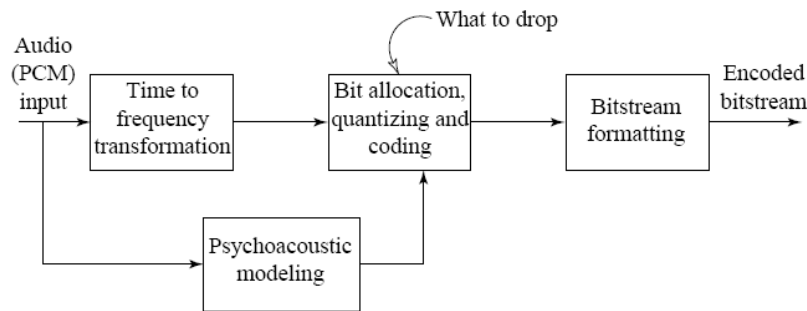


FIGURE 3 – Schéma d'encodage de la norme MPEG-1

Nous allons nous restreindre dans ce TP à étudier l'effet de l'allocation de bits en fréquence sur la qualité perceptive du codage réalisé à partir d'une simple analyse temps-fréquence par TFCT.

4.1 Codage audio perceptif

Nous allons commencer par encoder le signal audio d'entrée à partir d'un débit d'encodage souhaité.

1. Calculer la TFCT X du signal audio x (.wav de votre choix, avec une taille de fenêtre et un pas d'avancement judicieusement choisis en fonction de la fréquence d'échantillonnage du signal audio, et tracer le spectrogramme correspondant en fonction du temps en secondes et des fréquences en Hertz.
2. Pour chaque trame temporelle n , calculer le facteur de gain A_n permettant la normalisation unitaire du spectre d'amplitude¹.
3. Calculer, à partir d'un débit souhaité de 392 kbits/s, le nombre de bits alloué à chaque trame du module de la TFCT, noté $|X^{norm}|$, en fonction des paramètres d'analyses utilisés et de la fréquence d'échantillonnage. En déduire le nombre de bits alloués en moyenne à chaque point fréquentiel.
4. Calculer la répartition des bits en fréquence pour chaque trame, en utilisant l'algorithme d'allocation de bits perceptif qui minimise la différence entre le rapport signal à masque SMR calculé à partir d'un modèle psycho-acoustique et le rapport SNR calculé en fonction du nombre de bits utilisés pour la quantification. Pour rappel :

$$NMR(dB) = SMR(dB) - SNR(dB)$$

Pour simplifier, le SMR sera calculé avec un niveau de masque égal à -96 dB (16 bits maximum pour chaque point fréquentiel), et on utilisera l'hypothèse que l'allocation d'1 bit entraîne une augmentation du SNR de 6 dB (et donc une diminution du NMR de 6 dB). On en déduira une matrice d'allocation Q .

1. La normalisation unitaire consiste à normaliser à 1 la valeur maximale de chaque colonne de la TFCT.

5. Expliciter les conditions d'arrêts à utiliser pour l'arrêt de l'allocation.
6. En déduire le spectrogramme quantifié $|X_q^{norm}|$ à partir du nombre de bits donné par les éléments de la matrice Q . On utilisera pour cela la fonction **Fuquant** donnée dans le matériel du TP4. Tracer le spectrogramme et comparer avec le spectrogramme original en fonction du nombre de bits alloués. Quelles fréquences disparaissent majoritairement ? Expliquer.

Maintenant que nous avons notre signal compressé, nous souhaitons reconstruire le signal sonore correspondant pour pouvoir l'écouter.

7. A partir de la matrice de codage $|X_q^{norm}|$ et du nombre de bits alloués donné dans la matrice Q , restituer la valeur de chaque point fréquentiel. On utilisera la fonction de déquantification **Fuquant_inv** donnée dans le matériel du TP4. On pensera également à remettre à l'échelle les spectres d'amplitude des trames avec les facteurs de gains A_n . On stockera le résultat dans une matrice de codage déquantifiée et dénormalisée notée $|X_{uq}|$.
8. A partir de la matrice $|X_{uq}|$ reconstruire le signal audio compressé y à partir de sa TFCT en utilisant la fonction **tfct_inv**. On pensera pour cela à récupérer l'information de phase du signal initial, avant compression.
9. Comparer les signaux audio restitués après compression et décompression avec différents débits. Commenter l'impact du débit sur la compression.
10. A partir de quel débit d'encodage, la compression devient-elle transparente - c'est-à-dire que la différence entre le signal non-compressé et le signal compressé est inaudible ?
11. Conclure sur l'intérêt de l'allocation perceptive de bits par rapport à une allocation uniforme. Que faudrait-il faire pour rendre le codage plus efficace ?

4.2 BONUS

1. Recommencer et comparer avec une allocation uniforme de bits.
2. Réaliser l'allocation de bits par bandes de fréquences (par exemple par bandes de 32 points fréquentiels).
3. Comparer la compression avec différents enregistrements audio.
4. Participer à l'expérience sur la perception de la musique compressée disponible à l'adresse : <https://tinyurl.com/qembtrt>, et rapporter le score et les impressions.

Aide Python

```
import numpy as np
import matplotlib.pyplot as plt
print(C)
```

bibliothèque standard d'outils mathématiques
bibliothèque standard d'outils graphiques
affiche dans la console la valeur de la variable C

```
a=np.array([[1, 2, 3]])
a.T
A = np.array([[1, 2, 3],[4, 5, 6]])
A.T
```

Vecteur ligne $[1, 2, 3]$
Vecteur colonne $[1, 2, 3]^T$
Matrice de taille 2×3
Transpose la matrice

```
np.linspace(0, 10, 1000)
np.zeros((12, 5))
np.ones((12, 5))
np.eye(5)
```

Crée un vecteur de 1000 éléments espacés linéairement entre 0 et 10
Crée une matrice de taille 12×5 remplie de zéros
Crée une matrice de taille 12×5 remplie de uns
Crée une matrice identité de taille 5×5

```
x[1]
x[1:11]
x[1:]
x[:3]
```

Récupère le 2^{ème} élément de x
Récupère un vecteur composé du 2^{ème} au 12^{ème} élément de x
Récupère un vecteur composé du 2^{ème} au dernier élément de x
Récupère un élément sur 3 de x , en partant du premier jusqu'au dernier élément de x

```
A[1,0]
A[4,:]
A[:,4]
A[2, 1:4]
```

Récupère $A_{2,1}$
Récupère la 5^{ème} ligne de A
Récupère la 5^{ème} colonne de A
Récupère du deuxième au 4^{ème} élément de A dans la 3^{ème} ligne
Attention : dans $A[2, 1 :4]$, la variable 4 est exclusive (càd que les éléments retenus seront dans les colonnes 1,2,3)

```
A.dot(y)
A.dot(B)
A * B
np.linalg.matrix_power(A,3)
A**3
np.linalg.inv(A)
A.shape
```

Produit matriciel d'une matrice par un vecteur
Produit matriciel d'une matrice par une autre matrice
Produit élément-par-élément de deux matrices
Matrice A au cube (produit matriciel)
Mise au cube élément-par-élément de A
Matrice inverse de A
Taille de A

```
3**7
np.sqrt(5)
np.log(3)
np.log10(100)
np.abs(-5)
np.sin(5*pi/3)
np.floor(3.8)
```

Calcule 3^7
Calcule $\sqrt{5}$
Calcule $\ln(3)$
Calcule $\log_{10}(100)$
Calcule $|-5|$
Calcule $\sin(5\pi/3)$
Calcule la partie entière de 3.8

```
fig=plt.figure()
fig.plot(x,y)
fig.title('Un titre')
fig.xlabel('Grandeur x')
fig.legend('toto')
fig.grid()
plt.xlim((xmin, xmax))
plt.show()
```

création d'un environnement figure
Représente y en fonction de x (doivent être de la même taille)
Ajoute un titre à la figure
Ajoute une description à l'axe des abscisses
Ajoute une légende pour les courbes
Ajoute une grille à la figure
Choisit la limite pour l'axes des abscisses
Affiche les graphes

```
import scipy.io.wavfile as wav
wav.read('myfile')

from playsound import playsound
playsound('myfile.wav')
```

bibliothèque standard d'outils de traitement du son
Retourne le signal audio contenue dans `fichier` ainsi que sa fréquence d'échantillonnage
bibliothèque standard d'outils de traitement du son
Permet d'écouter le signal audio `myfile.wav`

```
np.hamming(L)
```

Retourne une fenêtre de Hamming de L points

```
np.pi
np.NaN
np.inf
plt.imshow
```

$\pi = 3.141592653589793$
Not a number, par exemple le résultat de $0/0$
Infinie
permet d'afficher un spectrogramme