# Compte-rendu de projet PPR1

Application MATLAB : Analyse et traitement du signal audio

---

Réalisé par : Lucie Perouas

Dans le cadre de la deuxième année de Licence Sciences de l'ingénieur parcours renforcé (CUPGE)

Sous la tutelle de Gilles Chabriel

Année universitaire : 2025 - 2026

# Table des matières

# 1 Introduction

Le traitement numérique du signal joue un rôle essentiel dans l'analyse et l'exploitation des données issues du monde physique. Il intervient notamment dans la conversion de signaux continus en représentations discrètes afin de permettre leur étude par des moyens informatiques. Cette étape de numérisation s'accompagne de nombreuses opérations fondamentales telles que le filtrage, l'analyse fréquentielle ou encore la réduction du bruit. On retrouve ces méthodes dans des applications concrètes comme les télécommunications (transmission et compression de données), le traitement audio (amélioration et synthèse sonore), ou encore l'imagerie médicale, où elles contribuent à l'interprétation précise des signaux biologiques. Pour un étudiant en ingénierie, cette discipline constitue ainsi un outil central permettant de relier les modèles mathématiques aux applications concrètes du traitement de l'information.

Dans ce contexte, ce projet personnel de recherche vise à initier au traitement du signal à travers la conception d'une application interactive développée sous MATLAB App Designer. Destinée à des étudiants de deuxième année de Sciences de l'ingénieur, cette application a pour objectif de rendre plus concrètes les notions fondamentales liées à la numérisation des signaux. Elle permet notamment de visualiser et d'analyser les effets de l'échantillonnage, de la quantification et du filtrage, afin de mieux comprendre leur impact sur la représentation et la qualité des signaux. L'application développée dans ce projet permet ainsi de manipuler directement des signaux synthétiques ainsi que des signaux audio réels, afin d'observer de manière visuelle et interactive l'impact des différents paramètres de traitement. L'utilisateur peut notamment étudier l'effet du nombre de bits sur la quantification, l'influence de la fréquence d'échantillonnage sur la représentation du signal, ainsi que les transformations associées au passage du domaine temporel au domaine fréquentiel.

Le choix de MATLAB App Designer s'explique par le fait que MATLAB est un environnement couramment utilisé en ingénierie pour le calcul numérique, le traitement du signal et la visualisation de données. Il permet de mettre en œuvre facilement des notions comme la transformée de Fourier, le filtrage ou encore l'échantillonnage. App Designer permet quant à lui de créer des interfaces graphiques interactives directement intégrées au code, ce qui facilite la mise en relation entre les manipulations de l'utilisateur et les traitements effectués sur les signaux. Il est à noter que ce projet a été réalisé dans un cadre étudiant, ce qui a conduit à privilégier la compréhension des mécanismes fondamentaux et la mise en œuvre des traitements plutôt qu'une optimisation logicielle avancée. L'application développée constitue ainsi une première version fonctionnelle, principalement orientée vers un usage pédagogique. Elle permet déjà d'illustrer les notions essentielles du traitement numérique du signal. Plusieurs améliorations restent néanmoins envisageables, notamment l'enrichissement des analyses proposées ainsi que l'optimisation de l'ergonomie. Ce travail constitue donc une base, ouvrant la voie à des évolutions futures vers un outil plus complet.

# 2 Présentation et architecture de l'application MATLAB

L'objectif de cette application est de proposer un environnement interactif destiné à des étudiants de deuxième année de Licence, afin de leur permettre d'observer concrètement les effets des opérations fondamentales telles que l'échantillonnage, la quantification et le filtrage sur la représentation temporelle, fréquentielle et sonore. Pour des raisons de praticité, l'application a été développée sous MATLAB App Designer, qui permet de concevoir des interfaces graphiques tout en intégrant les traitements numériques associés.

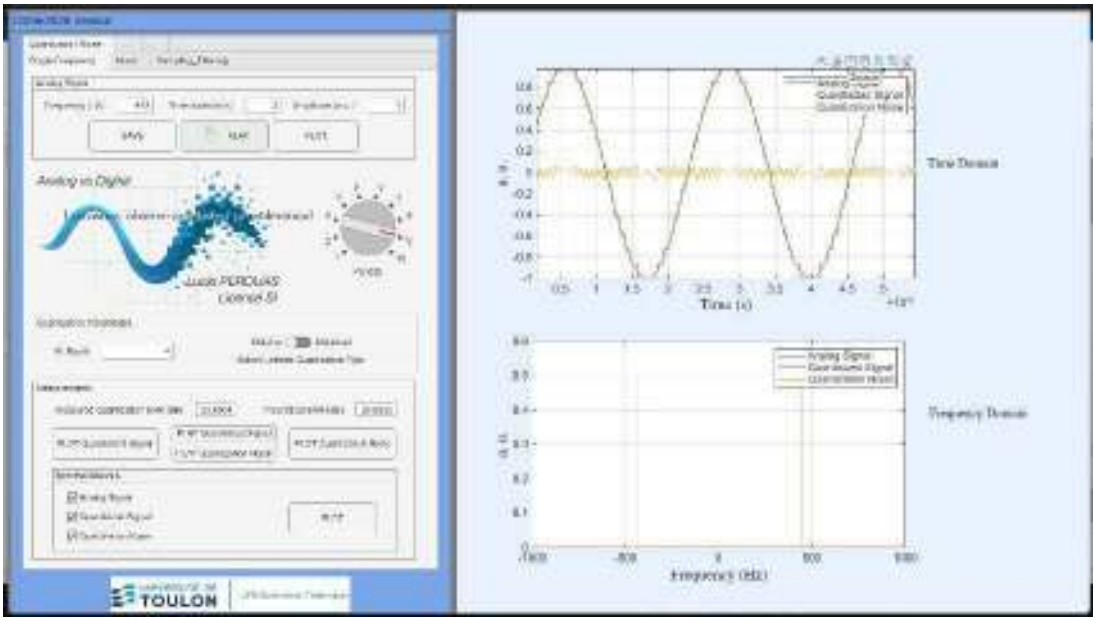


FIGURE 1 – Interface de l'application MATLAB

L'utilisateur lance l'application en exécutant en local le programme :

```
clc; close all;

global GlobalParameters
GlobalParameters.SoundCard_Fs = 44100;
GlobalParameters.temporary_save = './temporary_save/'; % save directory

projetPPR1;
```

Ce script initialise l'environnement de travail avant le lancement de l'application. La variable globale GlobalParameters est déclarée afin de partager des paramètres entre les différentes parties de l'application. La fréquence d'échantillonnage de la carte son est fixée à 44,1 kHz via GlobalParameters. un dossier de sauvegarde temporaire est défini avec GlobalParameters.temporarysave. La commande projetPPR1 lance l'application principale.

L'architecture de AppDesign repose sur une classe principale héritant de `AppBase`. Cette structure permet de centraliser l'ensemble des variables de l'application ainsi que les fonctions associées. Les interactions entre l'utilisateur et l'application sont gérées via des fonctions de rappel (callbacks), déclenchées automatiquement en fonction des actions

réalisées sur l'interface (clic sur un bouton, modification d'un paramètre, sélection dans un menu déroulant, etc.).

L'interface est organisée sous forme d'onglets afin de structurer les différentes fonctionnalités de manière progressive et intuitive. Chaque onglet correspond à une étape du traitement du signal, ce qui permet à l'utilisateur de suivre un chemin logique allant de la génération de signaux simples jusqu'à l'analyse de signaux audio réels.

Le premier onglet est dédié à la génération de signaux sinusoïdaux. L'utilisateur peut y définir les paramètres principaux du signal tels que la fréquence, l'amplitude, la durée ainsi que la phase initiale. Des boutons permettent ensuite de générer le signal, de l'afficher sous forme de courbe temporelle et de l'écouter directement via la carte son de l'ordinateur. L'utilisateur peut ajuster le nombre de bits de quantification. Le signal quantifié est affiché en parallèle du signal original afin de visualiser les différences introduites par la discrétisation. Des indicateurs numériques sont également affichés, notamment le rapport signal sur bruit (SNR), permettant d'évaluer la qualité de la quantification. En troisième partie de l'onglet "Simple Frequency", l'utilisateur peut visualiser le spectre du signal en utilisant une transformée de Fourier rapide. Différentes options permettent de comparer le spectre du signal original et celui du signal quantifié ainsi que le spectre du bruit de quantification.

Le deuxième onglet "Music" permet l'importation et le traitement de fichiers audio externes (formats WAV et MP3). L'utilisateur peut sélectionner un fichier via un explorateur intégré, puis le charger dans l'application. Une fois importé, le signal audio peut être visualisé dans le domaine temporel, analysé dans le domaine fréquentiel et écouté. L'utilisateur peut le quantifier et visualiser les effets de quantification dans le domaine fréquentiel comme pour l'analyse de la monofréquence.

Un troisième onglet Sampling/Filtering permet de rendre perceptible les effets de rééchantillonnage et de filtrage numérique. Dans la première partie, l'utilisateur peut importer un signal audio et modifier sa fréquence d'échantillonnage afin d'observer les effets du sous-échantillonnage. Le signal est ensuite décimé de manière simple, ce qui permet de visualiser les phénomènes de perte d'information et d'aliasing dans le domaine fréquentiel. Une comparaison des spectres du signal original et du signal rééchantillonné est affichée à l'aide de la transformée de Fourier rapide. Dans la deuxième partie, l'utilisateur peut appliquer différents types de filtres numériques (passe-bas, passe-haut, passe-bande et coupe-bande) en choisissant l'ordre du filtre ainsi que ses fréquences de coupure. Le signal filtré est ensuite affiché dans le domaine temporel, et la réponse fréquentielle du filtre est visualisée via une représentation pôles-zéros. Cela permet de relier directement les paramètres du filtre à son effet sur le signal.

# 3 Génération et manipulation d'un signal sinusoïdal

Le signal sinusoïdal constitue un élément fondamental en traitement du signal numérique. En effet, tout signal périodique peut être représenté comme une somme infinie de composantes sinusoïdales de fréquences harmoniques, conformément au théorème des séries de Fourier. Cette décomposition permet de passer du domaine temporel au domaine fréquentiel, ce qui facilite l'analyse et la compréhension des systèmes linéaires invariants dans le temps ainsi que des signaux plus complexes.

## 3.1 Modélisation mathématique du signal continu et discret

Un signal sinusoïdal continu est défini par :

$$x(t) = A\sin(2\pi ft + \varphi)$$

où $A$ représente l'amplitude, $f$ la fréquence et $\varphi$ la phase initiale.

Dans un système numérique, ce signal doit être discrétisé. Le temps continu $t$ est remplacé par un ensemble d'instants discrets $t = nT_e$, où $T_e = \frac{1}{f_e}$ est la période d'échantillonnage.

Le signal devient alors :

$$x[n] = A\sin\left(2\pi\frac{f}{f_e}n + \varphi\right)$$

Cette écriture met en évidence le rôle fondamental de la fréquence d'échantillonnage $f_e$, qui conditionne la fidélité de la représentation numérique.

Le théorème de Shannon-Nyquist impose la condition :

$$f_e \geq 2f_{\max}$$

permettant d'éviter les phénomènes de repliement spectral entraînant une déformation irréversible du signal.

## 3.2 Implémentation dans MATLAB App Designer

Dans l'application, la génération du signal sinusoïdal repose sur la synthèse numérique en temps discret. L'approche consiste à construire un vecteur temporel échantillonné à fréquence fixe, puis à évaluer l'expression mathématique d'une sinusoïde sur ce support discret. Les paramètres du signal sont directement fournis par l'utilisateur dans l'interface.

Le code est structuré en trois étapes principales.

Dans un premier temps, les paramètres globaux et les valeurs saisies par l'utilisateur sont récupérés :

```
SoundCard_Fs = GlobalParameters.SoundCard_Fs;
Freq = get(app.FrequencyHzEditField,'value');
Duration = get(app.TimedurationsEditField,'value');
Ampl = get(app.AmplitudeauEditField,'value');
```

Cette étape permet de définir la fréquence d'échantillonnage du système ainsi que les caractéristiques du signal à générer (fréquence, amplitude et durée).

Dans un second temps, les paramètres temporels du signal sont construits. Le pas d'échantillonnage est calculé à partir de la fréquence d'échantillonnage, puis le nombre total d'échantillons est déterminé en fonction de la durée du signal :

```
SamplingPeriod = 1/SoundCard_Fs;
NbSamples = Duration/SamplingPeriod;
t = [0:SamplingPeriod:(NbSamples-1)*SamplingPeriod];
```

Cette étape permet de créer un vecteur temps discret uniforme représentant l'axe temporel du signal.

Enfin, le signal sinusoïdal est généré en appliquant directement l'expression mathématique d'une sinusoïde :

```
SamplingPeriod = 1/SoundCard_Fs;
NbSamples = Duration/SamplingPeriod;
t = [0:SamplingPeriod:(NbSamples-1)*SamplingPeriod];
```

Le signal est ensuite généré point par point :

```
signal = Ampl*sin(2*pi*Freq*t);
```

Cette dernière étape réalise la synthèse du signal en temps discret à partir du vecteur temporel précédemment construit.

Puis le signal est affiché dans l'interface graphique à l'aide d'un callback dédié :

```
plot(app.UIAxes,t,signal)
grid(app.UIAxes,"on")
xlabel(app.UIAxes,'Time (s)')
ylabel(app.UIAxes,'Amplitude')
legend(app.UIAxes,'Analog Signal')
```

Enfin, le signal généré peut être interprété comme un signal audio et envoyé vers la carte son :

```
sound(signal, SoundCard_Fs)
```

Cette opération établit un lien direct entre représentation mathématique et perception auditive. La fréquence du signal est perçue comme la hauteur du son, tandis que l'amplitude correspond à son intensité sonore (ici fixée à 1). Cette première étape de l'application permet donc de relier trois représentations complémentaires du signal : la représentation mathématique, la représentation temporelle, et la perception auditive. Elle constitue une base essentielle pour la suite du projet, notamment pour l'étude de la quantification et de l'analyse fréquentielle.

# 4 Quantification uniforme du signal

## 4.1 Principe général de la quantification

La quantification constitue une étape fondamentale du traitement numérique du signal. Elle consiste à transformer un signal analogique, dont les amplitudes sont continues, en un signal discret en amplitude prenant un nombre fini de valeurs. Cette opération s'inscrit dans la chaîne de conversion analogique-numérique (A/N), qui comprend généralement trois étapes principales : l'échantillonnage du signal, la quantification de son amplitude et le codage binaire des valeurs obtenues.

Dans les systèmes numériques classiques, la quantification est souvent réalisée sur des résolutions standards comprises entre 8, 10, 12 ou 16 bits. Le nombre de bits détermine directement le nombre de niveaux de quantification disponibles : plus il est élevé, plus la représentation du signal est fine et fidèle à l'original analogique. À l'inverse, une faible résolution introduit une erreur de quantification plus importante, assimilable à un bruit ajouté au signal.

Dans l'application, la quantification est utilisée afin d'illustrer concrètement la perte d'information induite par la numérisation. Elle permet également d'analyser l'impact du nombre de bits sur la qualité du signal et sur le niveau de bruit de quantification introduit, ainsi que sur le rapport signal sur bruit (SNR). Cela met en évidence le compromis fondamental entre précision de représentation et complexité de codage dans les systèmes de traitement numérique du signal.

## 4.2 Implémentation de la quantification dans l'application

Dans l'application, la quantification est implémentée sous la forme d'une fonction dédiée :

```
function signalq = decreaseQuantization(signal,n,Ampl)

signalq = signal*(2^(n-1)-1)/Ampl;
signalq = round(signalq); % mid-rise
signalq = signalq/(2^(n-1)-1)*Ampl;
```

Cette fonction est appelée à plusieurs endroits dans l'application, notamment lors de la lecture du signal quantifié, de son affichage et du calcul du rapport signal sur bruit.

La première ligne :

$$signalq = signal \cdot \frac{2^{n-1} - 1}{Ampl}$$

correspond à une mise à l'échelle du signal. L'objectif est de ramener les amplitudes du signal dans un intervalle adapté au nombre de niveaux de quantification disponibles, défini par le nombre de bits $n$. Le facteur $(2^{n-1} - 1)$ correspond au nombre de niveaux positifs disponibles dans un codage uniforme. Le choix du type de quantification dans l'application (mid-rise ou mid-tread) peut ensuite influencer la position des niveaux de décision et la représentation du zéro, mais n'est pas directement visible dans cette étape de normalisation.

La commande :

```
signalq = round(signalq);
```

réalise la véritable opération de quantification. Chaque valeur continue est arrondie au niveau discret le plus proche. Cette opération introduit une erreur irréversible appelée erreur de quantification.

La dernière ligne :

$$signalq = \frac{signalq}{2^{n-1} - 1} \cdot Ampl$$

permet de remettre le signal dans la même plage d'amplitude que le signal original afin de pouvoir comparer directement les deux signaux sur les graphiques et lors de l'écoute audio.

La fonction de quantification est utilisée dans plusieurs callbacks de l'application, notamment lors de la lecture et de l'affichage du signal quantifié.

Par exemple, lors de la lecture audio du signal quantifié :

```
n = get(app.BitDepthEditField,'value'); % nombre de bits
signalq = decreaseQuantization(signal,n,Ampl);
sound(signalq,SoundCard_Fs)
```

De même, pour l'affichage temporel du signal quantifié :

```
signalq = decreaseQuantization(signal,n,Ampl);
plot(app.UIAxes,t,signalq)
```

Ces appels permettent d'observer directement l'effet du nombre de bits de quantification choisi par l'utilisateur, $n$ sur la qualité du signal.

## 4.3 Calcul du bruit de quantification

L'erreur introduite par la quantification est définie par la différence entre la valeur du signal x à l'échantillon n et la valeur du signal quantifié au même échantillon :

$$e_q[n] = x_q[n] - x[n]$$

Dans l'application, ce bruit est calculé par la commande suivante afin d'être visualisé.

```
noiseq = signalq - signal;
```

Cette erreur est ensuite analysée en termes de puissance afin de quantifier la dégradation du signal.

## 4.4 Calcul du rapport signal sur bruit (SNR)

Le rapport signal sur bruit est défini par :

$$\mathrm{SNR}_{\mathrm{dB}} = 10 \log_{10} \left( \frac{P_{\mathrm{signal}}}{P_{\mathrm{bruit}}} \right)$$

Dans l'application, les puissances des signaux sont estimées à partir de leurs variances empiriques. En effet, pour un signal centré (c'est-à-dire de moyenne nulle), la puissance moyenne est égale à la variance :

$$P_x = E[x^2] = \mathrm{Var}(x) \quad \text{si } E[x] = 0$$

Ainsi, le calcul numérique est réalisé comme suit :

```
powerSignal = std(signal-mean(signal))^2;
powernoiseq = std(noiseq-mean(noiseq))^2;
SNRqdB = 10*log10(powerSignal/powernoiseq);
```

Le bruit de qantification est ensuite visualisé ou écouté afin de percevoir l'impact du nombre de bits de quantification sur la qualité du signal. Plus le nombre de bits augmente, plus le pas de quantification diminue, et plus le signal quantifié se rapproche du signal original. À l'inverse, une faible résolution entraîne une augmentation significative du bruit de quantification.

# 5 Analyse spectrale du signal (Transformée de Fourier)

## 5.1 Principe général de la TFD

L'analyse fréquentielle permet de représenter un signal dans le domaine des fréquences afin d'en identifier les composantes principales. Contrairement au domaine temporel, qui décrit l'évolution du signal dans le temps, le domaine fréquentiel met en évidence les composantes spectrales du signal.Cette approche est essentielle en traitement du signal, car de nombreux phénomènes physiques sont plus simples à interpréter dans le domaine fréquentiel.

Le nombre d'échantillons d'un signal audio est donné par :

$$N = f_e \times T$$

où $f_e$ est la fréquence d'échantillonnage et $T$ la durée du signal.
Pour un fichier audio de 3 minutes ($T = 180\,s$) à $f_e = 44{,}100\,Hz$ :

$$N = 44100 \times 180 = 7\,938\,000 \text{ échantillons}$$

En pratique, un signal audio est analysé par blocs de taille finie (fenêtrage), et non sur toute sa durée. On utilise donc la Transformée de Fourier Discrète (TFD) sur des segments de $N' < N$ échantillons.

La TFD est définie par :

$$X[k] = \sum_{n=0}^{N'-1} x[n] e^{-j\frac{2\pi}{N'}kn}$$

Cette approche est nécessaire car un signal audio réel est non stationnaire, et l'analyse fréquentielle se fait localement dans le temps. Cette transformation permet de décomposer le signal en une somme de composantes fréquentielles complexes.
Le module $|X[k]|$ représente l'amplitude des différentes fréquences présentes dans le signal.

## 5.2 Implémentation de la FFT dans MATLAB App Designer

Dans l'application, l'analyse spectrale est réalisée via la transformée de Fourier rapide (FFT), qui est une version optimisée de la DFT.
Le code principal est le suivant :

```
signaltoPlot = fftshift(abs(fft(signaltoPlot)));
N = length(signaltoPlot);
Fe = 1/SamplingPeriod;
f = [-Fe/2:Fe/N:Fe/2-Fe/N];
```

La fonction `fft()` calcule la transformée de Fourier discrète du signal. Elle fournit un résultat complexe contenant amplitude et phase. Ici, seule l'amplitude est conservée à l'aide de la fonction `abs()` afin d'obtenir le spectre en module. Puis la fonction `fftshift()` est utilisée pour recentrer le spectre autour de zéro fréquence. Le centrage permet d'obtenir un spectre symétrique. L'axe fréquentiel est défini à partir de la fréquence d'échantillonnage :

$$f_e = \frac{1}{T_e}$$

Le vecteur fréquentiel est ensuite construit comme :

$$f \in \left[-\frac{f_e}{2}, \frac{f_e}{2}\right]$$

Cette construction permet d'associer chaque coefficient de la FFT à une fréquence physique en Hertz.

## 5.3 Gestion de l'affichage graphique dans l'application

L'application permet de comparer plusieurs spectres grâce à des cases à cocher. Chaque case permet d'afficher indépendamment : le spectre du signal analogique, le spectre du signal quantifié, et le spectre du bruit de quantification. Cela permet de faire une comparaison des différents signaux.

Le code permettant la gestion de l'affichage est le suivant :

```
if app.AnalogSignalCheckBox.Value == 1
    plot(app.UIAxes2,f,signaltoPlot/N)
    hold(app.UIAxes2,"on")
end

if app.QuantisizedSignalCheckBox.Value == 1
    signaltoPlot = signalq;
    signaltoPlot = fftshift(abs(fft(signaltoPlot)));
    plot(app.UIAxes2,f,signaltoPlot/N)
    hold(app.UIAxes2,"on")
end

if app.QuantizationNoiseCheckBox.Value == 1
    signaltoPlot = signalq - signal;
    signaltoPlot = fftshift(abs(fft(signaltoPlot)));
    plot(app.UIAxes2,f,signaltoPlot/N)
end
```

Lorsque la case `AnalogSignalCheckBox` est cochée, le spectre du signal analogique de référence est affiché dans l'espace graphique dédié (UIAxes2). Ce signal correspond au signal original avant toute opération de quantification.

Lorsque la case `QuantisizedSignalCheckBox` est cochée, le spectre du signal quantifié est calculé puis affiché.Cette possibilité permet de visualiser l'effet de la quantification sur son contenu fréquentiel du signal originel.

Enfin, lorsque la case `QuantizationNoiseCheckBox` est cochée, le bruit de quantification est calculé comme la différence entre le signal quantifié et le signal original $e[n] = x_q[n] - x[n]$. Son spectre est ensuite calculé et affiché afin d'analyser la distribution fréquentielle du bruit pur.

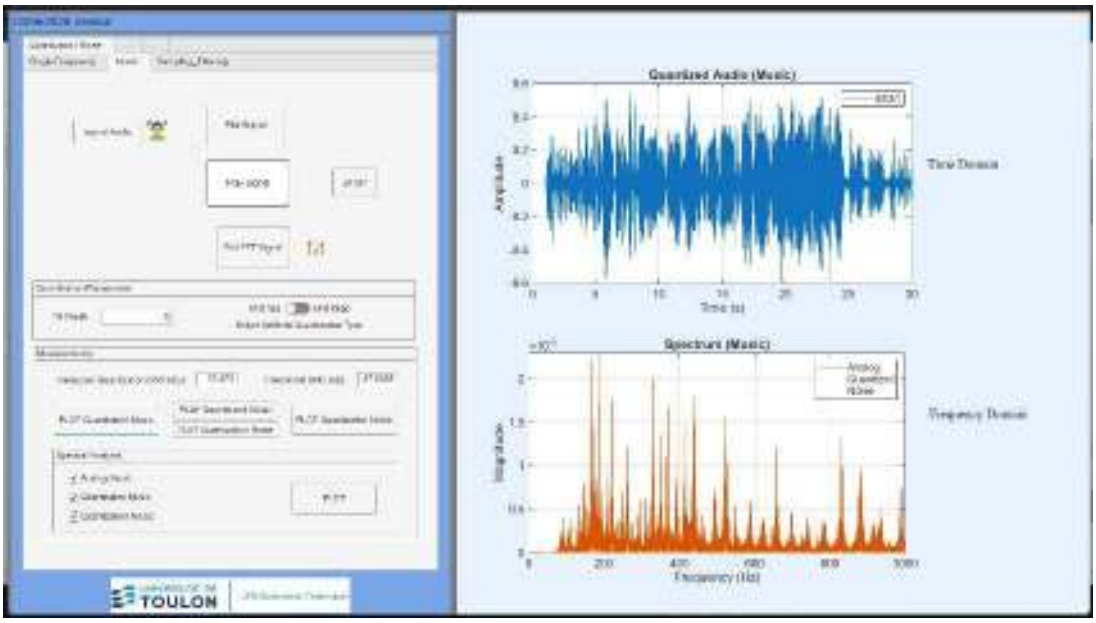


FIGURE 2 – Deuxième onglet "Music" de l'application MATLAB, permettant le traitement de fichiers mp3 ou wav

# 6 Traitement des signaux audio (musique)

Le traitement des signaux audio constitue une application directe des concepts fondamentaux du traitement du signal numérique. Contrairement aux signaux synthétiques (sinusoïdes), les signaux audio réels présentent une structure beaucoup plus complexe, composée d'un grand nombre de composantes fréquentielles.

L'objectif de cette partie de l'application est de permettre : l'importation de fichiers audio (mp3, wav), leur visualisation temporelle, leur lecture, et l'étude des effets de la quantification et de l'échantillonnage (partie suivante) sur un signal réel.

## 6.1 Importation et représentation numérique du signal audio

Un signal audio numérique est représenté sous la forme d'un vecteur discret $y[n]$ échantillonné à une fréquence $F_s$.

Dans l'application, l'importation est réalisée via l'appui sur le bouton "Import Audio" déclenchant l'explorateur de fichiers via l'exécution de la fonction suivante :

```
function ImportAudioButtonPushed2(app, event)
            [file, path] = uigetfile({'*.mp3;*.wav'});
            if file == 0
                return;
            end
            [y, Fs] = audioread(fullfile(path,file));

            if size(y,2) > 1
                y = mean(y,2);
            end
            app.AudioSignal = y;
            app.AudioFs = Fs;
            app.AudioTime = (0:length(y)-1)/Fs;
        end
```

La fonction `uigetfile()` ouvre une fenêtre de dialogue permettant à l'utilisateur de sélectionner un fichier audio au format `.wav` ou `.mp3`. Elle retourne le nom du fichier ainsi que son chemin d'accès.

La fonction `audioread()` assure ensuite la conversion du fichier audio en un signal numérique exploitable. Concrètement, elle lit le contenu du fichier et le transforme en un tableau de valeurs numériques. Chaque valeur du tableau correspond à l'amplitude du signal audio à un instant d'échantillonnage donné. Ainsi, le signal audio est représenté sous forme discrète : $y[n]$ est un vecteur contenant les échantillons successifs du signal dans le temps. La variable $F_s$ correspond à la fréquence d'échantillonnage du fichier audio, c'est-à-dire le nombre d'échantillons par seconde utilisés lors de la numérisation initiale du signal. Cette étape correspond donc à la transition entre un fichier audio et sa représentation numérique sous forme de vecteur de données exploitable pour le traitement du signal dans l'application.

A noter que les signaux audio peuvent être enregistrés en stéréo (deux canaux). Dans ce cas, le signal est représenté par une matrice à deux colonnes. Afin de simplifier le traitement, le signal est converti en mono :

```
if size(y,2) > 1
    y = mean(y,2);
end
```

C'est à dire que si le signal possède deux canaux (gauche et droite), une moyenne est effectuée afin d'obtenir un seul signal représentatif. Cette opération permet : de réduire la complexité du traitement, d'uniformiser les calculs de FFT et de quantification, et de simplifier l'analyse dans l'application.

## 6.2 Visualisation temporelle du signal audio

Le signal audio est ensuite affiché dans le domaine temporel. L'appui sur le bouton "Plot Signal" de l'interface déclenche la fonction Plot :

```
function PlotSignalButtonPushed(app, event)
            if isempty(app.AudioSignal)
```

```
            return;
        end

        Fs = app.AudioFs;
        y = app.AudioSignal;

        % limiter à 30 secondes
        maxSamples = min(length(y), round(30*Fs));
        y = y(1:maxSamples);

        t = (0:maxSamples-1)/Fs;

        plot(app.UIAxes, t, y)
        grid(app.UIAxes,"on")
        box(app.UIAxes,"on")

        xlabel(app.UIAxes,'Time (s)')
        ylabel(app.UIAxes,'Amplitude')
        title(app.UIAxes,'Audio signal (first 30 s)')

    end
```

Dans un premier temps, une vérification est effectuée afin de s'assurer qu'un signal audio a bien été chargé. Si la variable `app.AudioSignal` est vide, la fonction s'arrête immédiatement afin d'éviter toute erreur d'exécution.

```
if isempty(app.AudioSignal)
return;
end
```

Ensuite, les données audio sont récupérées depuis les variables de l'application. `Fs` correspond à la fréquence d'échantillonnage du signal, tandis que `y` contient les échantillons du signal audio sous forme de vecteur.

```
Fs = app.AudioFs;
y = app.AudioSignal;
```

Afin de faciliter la visualisation, le signal est limité aux 30 premières secondes. Le nombre d'échantillons correspondant est calculé à partir de la fréquence d'échantillonnage, puis le signal est tronqué en conséquence.

```
maxSamples = min(length(y), round(30*Fs));
y = y(1:maxSamples);
```

Un vecteur temporel est ensuite construit. Chaque échantillon est associé à un instant $t$ exprimé en secondes, en fonction de la fréquence d'échantillonnage.

```
t = (0:maxSamples-1)/Fs;
```

Enfin, le signal est affiché dans l'interface graphique. La courbe représente l'évolution de l'amplitude du signal en fonction du temps. Des éléments de mise en forme sont ajoutés afin d'améliorer la lisibilité du graphe (grille, cadre, labels et titre).

```
plot(app.UIAxes, t, y)
grid(app.UIAxes,"on")
box(app.UIAxes,"on")

xlabel(app.UIAxes,'Time (s)')
ylabel(app.UIAxes,'Amplitude')
title(app.UIAxes,'Audio signal (first 30 s)')
```

## 6.3 Lecture audio du signal

L'application permet la lecture audio du signal par appui sur le bouton "Play Signal", déclanchant la fonction suivante :

```
function PlaySignalButtonPushed(app, event)
            if isempty(app.AudioSignal)
                return;
            end

            sound(app.AudioSignal, app.AudioFs)

        end
```

Dans un premier temps, une vérification est effectuée afin de s'assurer qu'un signal audio a bien été chargé dans l'application. Si la variable `app.AudioSignal` est vide, la fonction s'arrête immédiatement afin d'éviter toute erreur d'exécution.

Ensuite, la fonction MATLAB `sound()` est utilisée pour restituer le signal audio via la carte son de l'ordinateur. Cette fonction prend deux arguments : le vecteur contenant les échantillons du signal audio, ainsi que la fréquence d'échantillonnage associée.

```
sound(app.AudioSignal, app.AudioFs)
```

La fréquence d'échantillonnage permet de garantir une restitution correcte du signal dans le domaine temporel, en respectant la vitesse de lecture originale. Cette étape permet ainsi à l'utilisateur d'écouter directement le signal importé au sein de l'application.

## 6.4 Analyse fréquentielle du signal audio

Bien que le signal audio soit plus complexe qu'un signal sinusoïdal, son analyse fréquentielle repose sur les mêmes principes que ceux introduits précédemment. L'affichage de la TFD du signal audio est déclenchée par l'appui sur le bouton "PLOT" du panneau "Spectral Analysis", déclanchant le callback suivant :

```
        function PlotFFTSignalButtonPushed(app, event)
            if isempty(app.AudioSignal)
                return;
```

```
        end

        Fs = app.AudioFs;
        y = app.AudioSignal;

        N = length(y);

        Y = fftshift(abs(fft(y)));
        f = linspace(-Fs/2, Fs/2, N);

        plot(app.UIAxes2, f, Y/N)
        grid(app.UIAxes2,"on")
        box(app.UIAxes2,"on")

        xlabel(app.UIAxes2,'Frequency (Hz)')
        ylabel(app.UIAxes2,'Magnitude')
        title(app.UIAxes2,'Spectrum')


    end
```

Après vérification de la présence d'un signal, celui-ci est récupéré ainsi que sa fréquence d'échantillonnage $F_s$. Le spectre est ensuite calculé à l'aide de la transformée de Fourier rapide (`fft`), puis recentré autour de la fréquence nulle grâce à `fftshift` afin d'obtenir une représentation symétrique. Un axe fréquentiel est construit sur l'intervalle $[-F_s/2, F_s/2]$ et le module du spectre est normalisé par le nombre d'échantillons $N$. Enfin, le résultat est affiché sous forme de courbe représentant l'amplitude du signal en fonction de la fréquence, permettant d'analyser son contenu spectral.

## 6.5 Quantification du signal audio

Le signal audio peut également être dégradé artificiellement par quantification afin d'étudier l'impact du nombre de bits sur la qualité sonore. L'utilisateur entre la valeur n du nombre de bits de quantification dans le champ "Bit Depth" du panneau "Quantizatin Parameters", déclanchant le callback :

```
function BitDepthEditField_2ValueChanged(app, event)
        if isempty(app.AudioSignal)
            return;
        end

        y = app.AudioSignal;

        n = app.BitDepthEditField_2.Value;
        A = max(abs(y));

        signalq = decreaseQuantization(y, n, A);

        % puissance signal / bruit
```

```
        powerSignal = std(y - mean(y))^2;
        noise = signalq - y;
        powerNoise = std(noise - mean(noise))^2;

        SNRdB = 10*log10(powerSignal / powerNoise);

        set(app.MeasuredQuantizationSNRdBTextArea_2,'Value',num2str(SNRdB))

        % théorique
        q = 1/(2^(n-1)-1);
        SNRTheoreticaldB = 10*log10(6/q^2);

        set(app.theoreticalQuantizationSNRdBTextArea_2,'Value',num2str(SNRTheoreti
    end
```

Le signal audio est d'abord récupéré, puis quantifié à l'aide de la fonction "decreaseQuantization" décrite plus haut. Le bruit de quantification est obtenu par différence entre le signal quantifié et le signal original. Les puissances du signal et du bruit sont ensuite estimées à partir de leur variance afin de calculer le rapport signal sur bruit (SNR) en dB. Ce SNR est affiché dans l'interface, en parallèle de sa valeur théorique calculée à partir du pas de quantification $q$, permettant ainsi de comparer performance pratique et modèle théorique.

# 7 Échantillonnage, rééchantillonnage

## 7.1 Théorie de l'échantillonnage (Shannon-Nyquist)

Le passage d'un signal continu à un signal discret repose sur l'échantillonnage :

$$x[n] = x(nT_e), \quad T_e = \frac{1}{f_e}$$

Le théorème de Shannon impose une condition essentielle :

$$f_e > 2f_{\max}$$

où $f_{\max}$ est la plus haute fréquence contenue dans le signal.

Si cette condition n'est pas respectée, un phénomène de repliement spectral apparaît, ce qui entraîne une mauvaise attribution des fréquences dans le spectre discret. Concrètement, des hautes fréquences sont repliées vers des fréquences plus basses, cela déforme le signal et rend impossible la reconstruction fidèle du signal original à partir de ses échantillons.

## 7.2 Sous-échantillonnage et repliement spectral

Dans cette partie, l'objectif est d'étudier l'impact du sous-échantillonnage sur un signal audio et d'illustrer expérimentalement le non respect du critère de Shannon. En réduisant artificiellement la fréquence d'échantillonnage, on cherche à observer comment la perte d'information dans le domaine temporel se traduit par une déformation du contenu fréquentiel, et dans quelles conditions la reconstruction fidèle du signal devient impossible.

Dans l'application, l'utilisateur choisit la fréquence de ré-échantillonnage dans le panneau "Resampling". Le sous-échantillonnage est simulé par une décimation simple :

```
y_ds = y(1:M:end);
```

Cette opération consiste à conserver un échantillon sur $M$, ce qui revient à réduire artificiellement la fréquence d'échantillonnage. Lorsque $f_s'$ devient inférieur à $2f_{\max}$, le spectre se replie et les composantes fréquentielles se superposent, ce qui dégrade fortement le signal. Cette implémentation permet d'illustrer directement les effets de l'aliasing sans filtrage anti-repliement préalable.

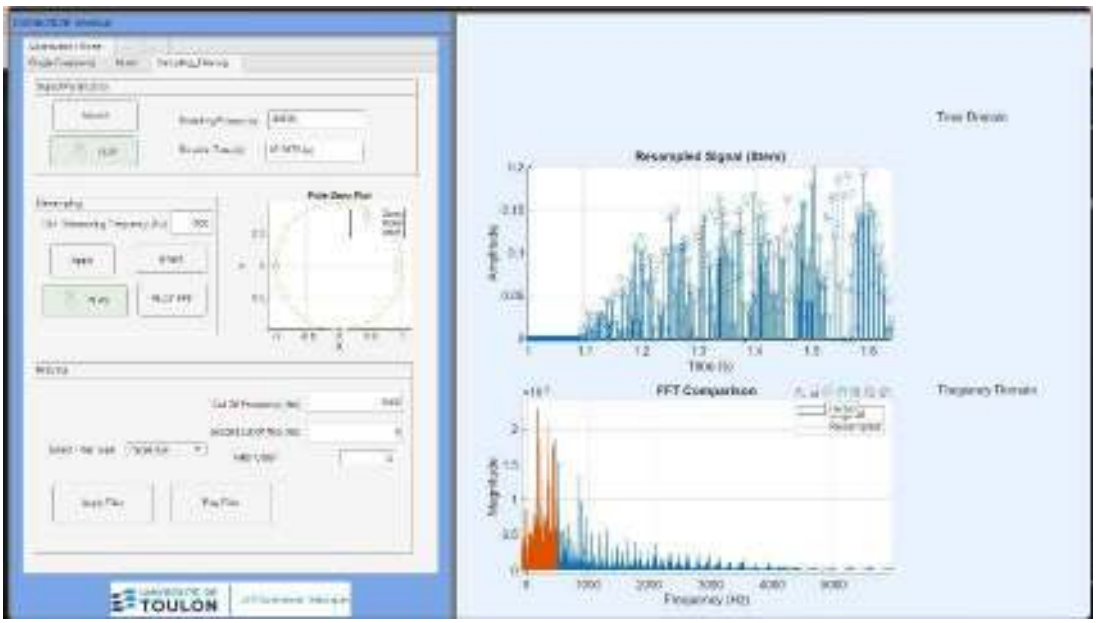


FIGURE 3 – Troisième onglet "Sampling / Filtering" de l'application MATLAB

## 7.3 Analyse fréquentielle (FFT)

Pour comparer les effets du rééchantillonnage, une analyse spectrale est réalisée via la transformée de Fourier discrète :

```
Y = fftshift(abs(fft(y)));
f1 = linspace(-Fs/2, Fs/2, N1);

Yr = fftshift(abs(fft(yr)));
f2 = linspace(-Fsr/2, Fsr/2, N2);
```

Le tracé comparatif est ensuite réalisé :

```
plot(app.UIAxes2, f1, Y/N1)
plot(app.UIAxes2, f2, Yr/N2)
legend(app.UIAxes2, "Original", "Resampled")
```

Ce résultat permet de visualiser directement la compression du spectre et les effets de repliement en cas de sous-échantillonnage.

# 8 Filtrage numérique

Lorsque des composantes de hautes fréquences sont présentes dans un signal et que la condition de Shannon n'est pas respectée, celles-ci peuvent se replier dans la bande des basses fréquences et venir perturber l'information utile. Afin de limiter cet effet, on distingue deux types de filtrage : le filtrage physique et le filtrage numérique. Le filtrage physique est réalisé en amont de la numérisation, directement sur le signal analogique, à l'aide de circuits électroniques (par exemple des filtres passe-bas analogiques). Il permet de supprimer les hautes fréquences avant l'échantillonnage et constitue donc une étape essentielle pour éviter l'aliasing à la source. Le filtrage numérique, quant à lui, est appliqué après la conversion du signal en données discrètes et agit directement sur les échantillons. Il permet de modifier le contenu fréquentiel du signal, soit pour supprimer des composantes indésirables, soit pour isoler une bande de fréquences utile. Dans ce contexte, les filtres de Butterworth sont utilisés en raison de leur réponse fréquentielle lisse et sans ondulation dans la bande passante, ce qui permet une atténuation progressive des hautes fréquences tout en préservant la qualité du signal dans la bande utile.

Cette fonctionnalité présente un intérêt direct pour le traitement de la parole, notamment après acquisition via microphone, en permettant de réduire les composantes haute fréquence parasites et le bruit ajouté lors de la numérisation, afin d'obtenir un signal plus propre et plus exploitable pour l'analyse ou la reconnaissance vocale.

## 8.1 Implémentation MATLAB du filtrage

Dans l'application, le filtrage est réalisé dans le callback suivant :

```
type = app.SelectFilterTypeDropDown.Value;
order = app.FilterOrderEditField.Value;

f1 = app.CutOffFrequencyHzEditField.Value;
f2 = app.SecondcutofffreqHzEditField.Value;

switch type
    case "Passe bas"
        Wn = f1/(Fs/2);
        [b,a] = butter(order, Wn, 'low');
    case "Passe haut"
        Wn = f1/(Fs/2);
        [b,a] = butter(order, Wn, 'high');
    case "Passe bande"
        Wn = [f1 f2]/(Fs/2);
        [b,a] = butter(order, Wn, 'bandpass');
    case "Coupe bande"
        Wn = [f1 f2]/(Fs/2);
        [b,a] = butter(order, Wn, 'stop');
end

y_f = filtfilt(b,a,y);
```

Ce code commence par récupérer les paramètres définis par l'utilisateur dans l'interface graphique, à savoir le type de filtre souhaité, l'ordre du filtre ainsi que les fréquences de coupure. Ces valeurs sont ensuite normalisées par rapport à la fréquence d'échantillonnage $F_s$ afin de respecter le domaine fréquentiel discret. En fonction du type de filtre sélectionné (passe-bas, passe-haut, passe-bande ou coupe-bande), la fonction `butter` de MATLAB calcule les coefficients du filtre de Butterworth correspondants. Le filtre est appliqué via `filtfilt`, ce qui présente un avantage majeur : l'absence de déphasage, car le filtrage est effectué en avant et en arrière sur le signal.

## 8.2 Visualisation des pôles et zéros

Une analyse supplémentaire est effectuée via la représentation pôles-zéros :

```
zplane(app.UIAxes2, b, a)
```

Cette représentation permet de vérifier la stabilité du filtre et d'interpréter son comportement fréquentiel.

# 9 Reconstruction d'un signal à partir de ses échantillons

La reconstruction d'un signal correspond au processus consistant à retrouver une approximation continue à partir de ses échantillons discrets $x[n]$. Cette étape est fondamentale en traitement du signal, car les systèmes numériques ne manipulent que des données discrètes, alors que les signaux physiques sont continus. La reconstruction d'un signal continu à partir de ses échantillons discrets $x[n]$ s'exprime idéalement par :

$$x_r(t) = \sum_{n=-\infty}^{+\infty} x[n] \operatorname{sinc}\left(\frac{t - nT_e}{T_e}\right)$$

où $T_e = \frac{1}{f_s}$ est la période d'échantillonnage et $\operatorname{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$.

## 9.1 Implémentation dans l'application

Dans le cadre de ce projet, cette fonctionnalité n'a pas encore été implémentée. Cependant, la fonctionnalité pourrait être intégrée sous la forme d'un onglet "signal reconstruction" et permettrait de comparer le signal original et les signaux reconstruits à partir des différentes modifications apportées (rééchantillonnage, quantification, filtrage numérique). L'utilisateur pourrait ainsi visualiser les effets du sous-échantillonnage et de l'aliasing, et tester différentes méthodes de reconstruction afin d'observer leur impact sur la qualité du signal. L'appui sur un bouton "APPLY Interpolation" déclancherait le callback :

```
function ApplyInterpolationButtonPushed(app, event)

    if isempty(app.AudioSignal)
        return;
    end
```

```
    % Signal discret
    x = app.AudioSignal;
    Fs = app.AudioFs;
    Te = 1/Fs;
    N = length(x);
    n = 0:N-1;

    % Temps continu (grille fine pour affichage)
    t = linspace(0, (N-1)*Te, 20*N);

    % Reconstruction de Shannon (interpolation sinc)
    x_recon = zeros(size(t));

    for k = 1:N
        x_recon = x_recon + x(k) * sinc((t - n(k)*Te)/Te);
    end

    % Affichage
    plot(app.UIAxes, n*Te, x, 'o'); % signal discret
    hold(app.UIAxes, "on");
    plot(app.UIAxes, t, x_recon);    % signal reconstruit
    hold(app.UIAxes, "off");
    grid(app.UIAxes, "on");
    box(app.UIAxes, "on");
    xlabel(app.UIAxes, 'Time (s)');
    ylabel(app.UIAxes, 'Amplitude');
    title(app.UIAxes, 'Signal Reconstruction (Shannon interpolation)');

end
```

Le code récupère le signal audio et sa fréquence d'échantillonnage, puis génère une grille temporelle fine afin d'approcher un comportement continu par interpolation de Shanon.

# 10 Conclusion

Ce projet a permis de développer une application complète de traitement du signal sous MATLAB App Designer, regroupant l'ensemble des étapes fondamentales du traitement numérique. L'application met en œuvre de manière interactive les principaux concepts étudiés en cours, notamment la génération de signaux sinusoïdaux et leur représentation temporelle, la quantification uniforme et l'analyse du bruit de quantification, l'analyse fréquentielle via la transformée de Fourier (FFT), le filtrage numérique ainsi que l'interpolation de reconstruction. L'ensemble de ces fonctionnalités permet de relier directement les modèles mathématiques aux effets observables dans les domaines temporel et fréquentiel, tout en offrant une approche intuitive et pédagogique du traitement du signal. Ce projet constitue ainsi une base solide pour l'étude de systèmes plus avancés tels que les traitements adaptatifs, la compression audio ou encore l'analyse spectrale approfondie.