PROJET SIGNAL

Accordeur électronique de guitare avec MATLAB

Colin Mourard <colin.mourard@ensica.isae.fr>
Anthony Pagliaï <anthony.pagliai@ensica.isae.fr>

ISAE - Formation ENSICA - Année 2013-2014

Table des matières

Première partie

Analyse de signaux synthétiques

Chapitre 1

Introduction

L'objectif de ce projet est de créer un accordeur électronique de guitare à l'aide du logiciel MATLAB. Ce projet est découpé en 2 parties. Ce rapport traite de la première partie dont l'objectif est de réaliser une analyse sur des signaux synthétiques.

Parlons un peu de musique ... Une guitare est composée de 6 cordes qui sont dimensionnées pour vibrer à une fréquence fondamentale donnée. Il convient donc d'accorder régulièrement ces 6 cordes pour préserver l'harmonie des sons. Chaque son est composée de plusieurs fréquences : une que l'on appelle la **fondamentale** et les autres qui sont les **harmoniques**. Le but de l'accordeur électronique est donc de permettre à n'importe quel musicien d'accorder chaque corde à la bonne fréquence (la fondamentale) sans pour autant être doté d'une oreille musicale...

En musique, l'unité utilisée est l'octave. Chaque octave comporte 6 tons eux-mêmes divisés en 2 demi-tons. Passer d'une octave n à l'octave n+1 pour une même note revient à doubler sa fréquence. Dans l'autre sens, la fréquence est divisée par 2.
Pour faciliter la manipulation des intervalles de fréquences, nous allons utiliser une unité plus appropriée qui est le **Savart**. La conversion nous est donnée par la relation suivante :

$$x_{sav} = 1000.log_{10}(x_{lin})$$

Ainsi, un octave correspond à environ 300 Savarts (en réalité, on a 1 octave $= 1000.log_{10}(2)$).

Hypothèse Nous supposerons que chaque corde est désaccordée au maximum à plus ou moins un quart de ton.

Chapitre 2

Questions

Tous les codes réalisés avec MatLab se trouvent dans l'***Annexe A***.

2.1 Construction d'un vecteur par concaténation

L'énoncé nous donne le vecteur e. Notre but est, à partir de ce vecteur, de retrouver un nouveau vecteur dont les composantes sont les fréquences fondamentales des 6 cordes à vide d'une guitare. Ce tableau est également donné dans l'énoncé.

En fait, il s'agit d'une simple conversion. La relation entre les écart des tons (composantes du vecteur e) et les fréquences fondamentales (composantes du vecteur f dans l'*Annexe A*) sont liés par une relation linéaire. Dans notre code, nous avons le **concept de vectorisation** qui permet de calculer plus vite sous MatLab qu'en utilisant une classique *boucle for*. Pour cette relation, nous obtenons :

$$f = 110.log_{10}(\frac{log_{10}(2)}{6}.e)$$

De plus, un ton correspond à $\frac{1}{6}$ d'octave et un demi-ton correspond à $\frac{1}{12}$ d'octave, donc en Savart nous avons les valeurs suivantes :

$$1\ ton \approx 50\ Savart$$

$$\frac{1}{2}\ ton \approx 25\ Savart$$

2.2 Créer une fonction MatLab

Le but de cette question est de nous sensibiliser à la création des fonctions en MatLab. Pour éviter toute erreur, ceci se fait toujours en 2 étapes :

1. Ecrire le script de la fonction dans un M-file de type *script*.
2. Une fois que nous sommes certains de notre code, créer un M-file de type *fonction* et recopier le script dans ce fichier.

N.B. Je n'ai mis dans l'*Annexe A* que le code associé au M-file de la fonction terminée.

La création de fonctions nous permet par la suite de faire directement appel à elles dans la *Command Window*. Plus concrètement, en ce qui nous concerne, les deux fonctions à coder nous serviront à convertir plus rapidement des données de la valeur en Savart à la valeur linéaire. Pour cela, il suffit d'utiliser la relation de passage donnée dans l'introduction.

2.3 Création et représentation du contenu d'un signal

Dans cette question, nous voulons créer un signal possédant une fondamentale de 110 Hz pendant une durée de 5 sec. L'échantillonage de ce signal impose le respect du *théorème de Shannon-Nyquist*

Théorème de Shannon-Nyquist
La fréquence d'échantillonage doit être choisie de telle sorte qu'il n'y ait pas de chevauchement des différentes composantes spectrales. Pour ce faire, la condition donnée par le théorème de Shannon-Nyquist impose :

$$f_s \geq 2.f_0$$

En pratique, nous choisirons cependant de sur-échantillonner notre signal pour avoir un signal le plus fidèle possible. Cependant il ne s'agit pas de saturer la mémoire de MatLab en calculant trop de points. Il faut trouver un bon compromis, qui sera, pour nous, de choisir une fréquence d'échantillonnage

$$f_s = 10.f_0 = 1100\ Hz$$

Représentation du signal échantillonné
Ci-dessous (FIGURE 2.1.) le graphe du signal de fréquence 110 Hz.

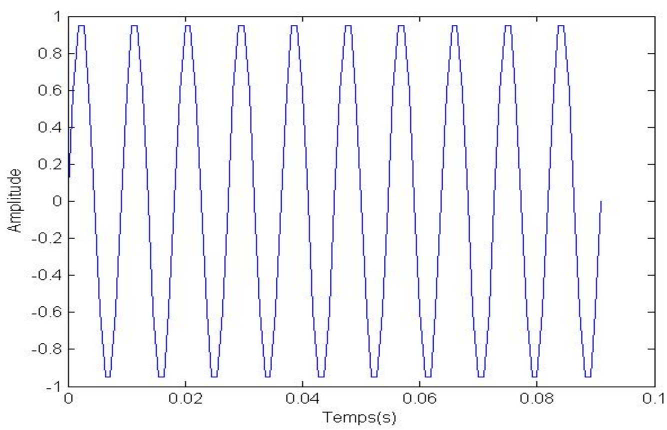

FIGURE 2.1 – Représentation du signal échantillonné sur 10 périodes

Analyse fréquentielle du signal par transformée de Fourier

La transformée de Fourier discrète L'expression analytique de la transformée de Fourier discrète (DFT) est donnée par :

$$S(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{2ik\pi n}{N}} \quad pour \quad 0 \leq k \leq N-1$$

Cependant, MatLab est un outil mathématique créé pour travailler avec des matrices. Nous pouvons travailler la relation analyique de la DFT pour obtenir une relation sous forme matricielle. Ainsi, le vecteur S dont chaque élément est une somme *S(k)* peut être obtenu par une multiplication de matrices, ce qui fait appel au fameux **concept de vectorisation**. Ainsi, on obtient la relation :

$$S = F^H x$$

où x est le vecteur colonne d'entrée et F une matrice carrée d'exponentielles telle que :

$$F = \left(f_{k,n}\right)_{(k,n)\in\{0\cdots N-1\}} \quad avec \quad f_{k,n} = e^{\frac{2i\pi kn}{N}}$$

Implémentation de la transformée de Fourier discrète Dans un premier temps, nous choisissons d'implémenter directement la relation matricielle que nous venons de dégager : $S = F^H x$. Pour cela, nous créons une fonction `DFT` qui calcule la DFT d'un signal donné en entrée. Le code associé à cette fonction est donné en annexe. Pour centrer le graphe en zéro, nous utilisons la fonction `fftshift`. Nous obtenons le graphique de la FIGURE 2.2., qui correspond bien à la transformée de Fourier d'un sinus de fréquence 110 Hz :

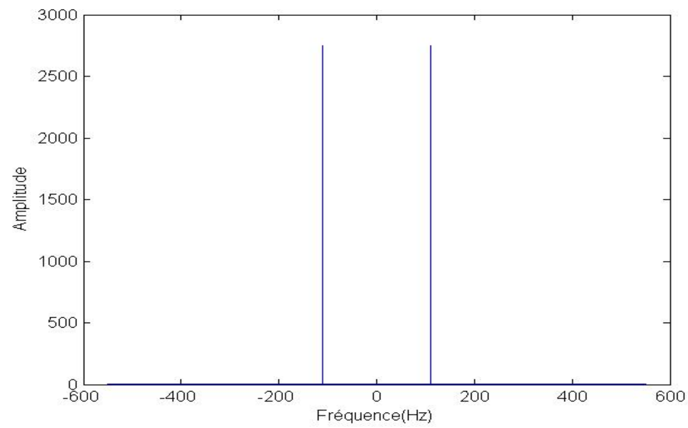

FIGURE 2.2 – Transformée de Fourier discrète du signal sinusoïdal de fréquence $f_0 = 110$ Hz

Ensuite, nous utilisons un algorithme rapide FFT pour représenter le contenu du signal d'entrée. Pour cela, nous utilisons la fonction `fft` implémentée par défaut en MatLab. De même que pour le premier graphe, la fonction `fftshift` nous permet de centrer le graphe en zéro.
Nous obtenons le graphique de la FIGURE 2.3. :

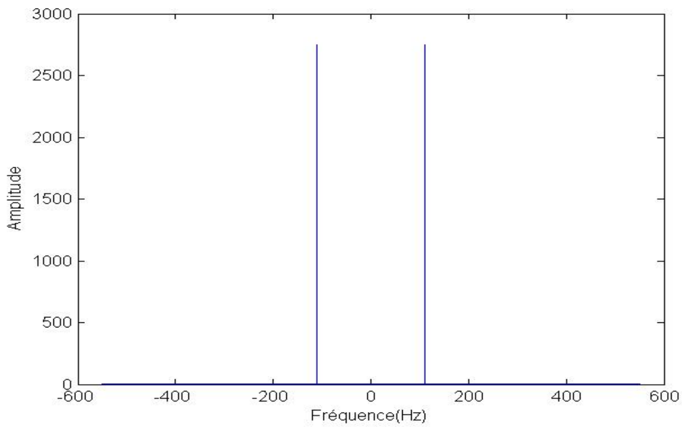

FIGURE 2.3 – Transformée de Fourier discrète du signal sinusoïdal de fréquence $f_0 = 110$ Hz

Chercher une référence bibliographique Nous devons chercher 3 informations différentes sur Internet.

- Un article publié en 1965 par Cooley-Tukey a permis de faire un grand pas en avant pour l'implémentation rapide de la DFT. Le nom de cet article est *An algorithm for the machine calculation of complex Fourier series.*
- Sur les archives ouvertes de Toulouse (OATAO) on peut trouver 26 **articles** comportant le mot *FFT*. Les domaines d'application associés sont **la médecine, l'aéronautique, l'optique, la robotique, les mathématiques ainsi que la localisation GPS**.
- Enfin, en ce qui concerne les archives ouvertes Arxiv, on compte 287 **articles** comportant le mot *FFT* dans leur résumé.

Précision de l'analyse par transformée de Fourier
Nous calculons analytiquement la transformée en Z de la séquence $[x]_m = [e^{j2\pi\overline{f_0}m}]_{m=0,\ldots,M-1}$ où f_0 est la fréquence normalisée, i.e. $\overline{f_0} = \frac{f_0}{f_s}$.
On ne s'intéresse qu'aux points $z = e^{j2\pi\overline{f}m}$ pour $\overline{f} = [0, 1]$.
Rappelons la formule de la transformée en Z d'une séquence de points $s(n)$:

$$T[s(n)] = \sum_{n=-\infty}^{+\infty} s(n)z^{-n}$$

Ainsi, en notant $X(Z)$ la transformée en Z de notre séquence $[x]_m$, nous obtenons :

$$\begin{aligned} X(Z) &= \sum_{m=0}^{M-1} e^{j2\pi \overline{f_0} m} z^{-m} \\ &= \sum_{m=0}^{M-1} e^{(j2\pi \overline{f_0} - ln(z))m} \ (suite\ géométrique) \\ &= \frac{1 - e^{(j2\pi \overline{f_0} - ln(z))M}}{1 - e^{(j2\pi \overline{f_0} - ln(z))}} \end{aligned}$$

Avec $z = e^{j2\pi \overline{f}}$ on a : $e^{ln(z)} = z = e^{j2\pi \overline{f}}$, d'où :

$$X(Z) = \frac{1 - e^{j2\pi(\overline{f_0} - \overline{f})M}}{1 - e^{j2\pi(\overline{f_0} - \overline{f})}}$$

Pour faire l'analyse de la DFT, nous créons donc une nouvelle fonction qui nous donne la transformée en Z d'une séquence x de points de fréquence f_0 lorsqu'on s'intéresse aux points $z = e^{j2\pi \overline{f}}$. Ensuite, nous représentons sur une même figure la DFT ainsi que la transformée en Z pour la séquence de points $\overline{f} = \texttt{linspace}(0, 1, 3e3)$. Cette fonction nous permet de créer une séquence de 3000 points régulièrement espacés et pris dans l'intervalle $[0, 1]$. Nous obtenons le graphe de la FIGURE 2.4. sur lequel on ne distingue pas les pics pour la transformée en Z :

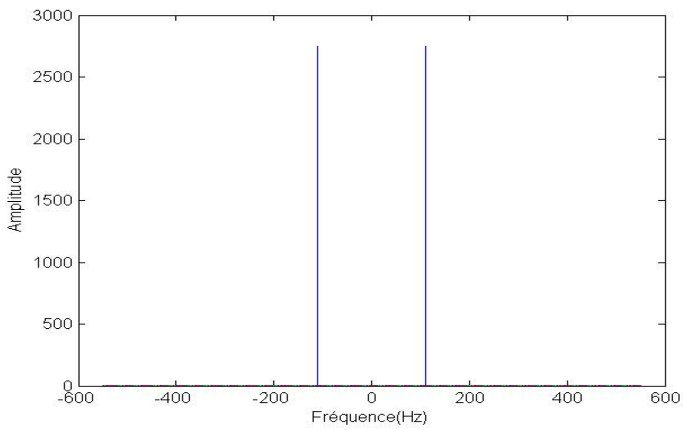

FIGURE 2.4 – Représentation de la DFT et de la transformée en Z

On souhaite maintenant augmenter la précision des représentations. Rappelons que la précision est donnée par la relation $\frac{f_s}{M}$ où M est le nombre de points calculés dans la somme de la DFT. Pour cela, nous pouvons utiliser la méthode du **zéro-padding**. Cela consiste à ajouter un très grand nombre de zéros à la fin de la séquence $[x]$ (par exemple, 50 ou 100 la longueur initiale du vecteur). Cela ne modifie pas la résolution du signal puisque celle-ci est déterminée par la durée du signal T, mais cela rajoute des points sur la transformée de Fourier que l'on obtiendra donc quasi-continue.

Résolution de l'analyse par transformée de Fourier La résolution du système au sens du critère de Rayleigh est en $\frac{1}{T}$ soit dans notre cas en 0.2 Hz. On trace alors les simulations numériques pour des écarts de fréquence $\Delta f \in \{1; 0.5; 0.2\}$. Les FIGURES 2.5., 2.6. et 2.7. récapitules ces résultats

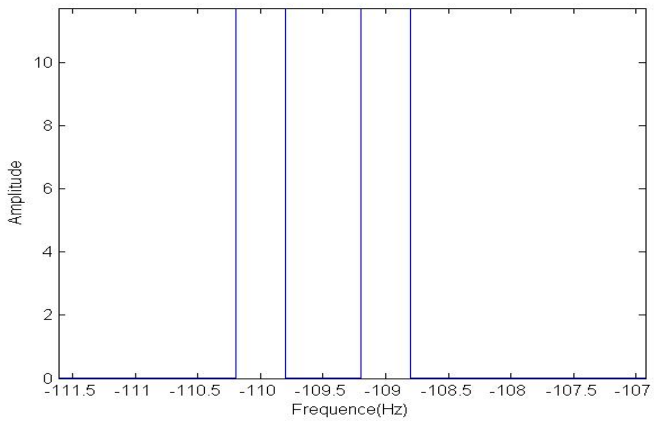

FIGURE 2.5 – Résolution pour $\Delta f = 1$ Hz

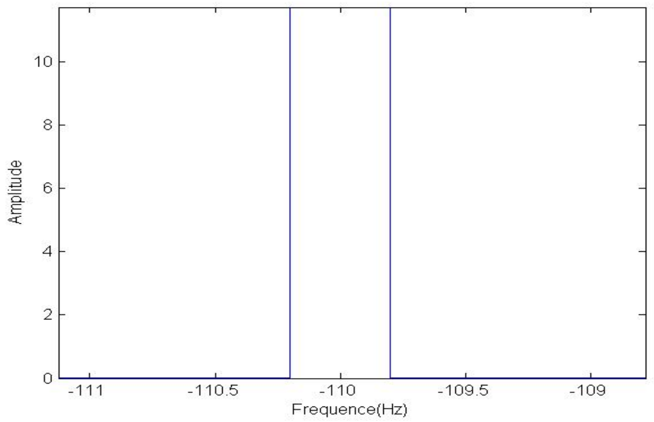

FIGURE 2.6 – Résolution pour $\Delta f = 0.5$ Hz

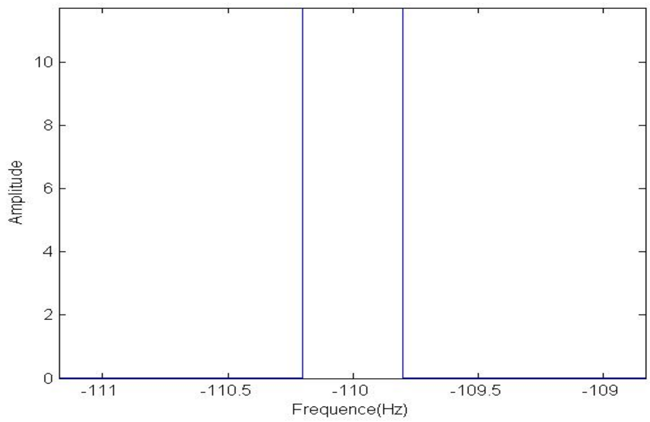

FIGURE 2.7 – Résolution pour $\Delta f = 0.2$ Hz

Notons que deux pics apparaissent sur la FIGURE 2.5. mais pas sur les FIGURES 2.6. et 2.7.. Cela signifie que l'écart de fréquence Δf est trop faible dans les 2 derniers cas pour être distingué lors du calcul de la DFT. Ils ne sont donc pas représentés.

Illustration du phénomène de repliement

Pour illustrer le phénomène de repliement, nous devons choisir une fréquence d'échantillonnage de manière à ne plus respecter le théorème de Shannon-Nyquist, soit $f_{s_{rep}} < 2.f_0$. Nous avons choisi de présenter 2 exemples, le premier pour une fréquence d'échantillonnage égale à $1.5.f_0$ et le second pour une fréquence d'échantillonnage égale à f_0. Les représentations de ces signaux sont données par les FIGURES 2.8. et 2.9. ci-dessous :

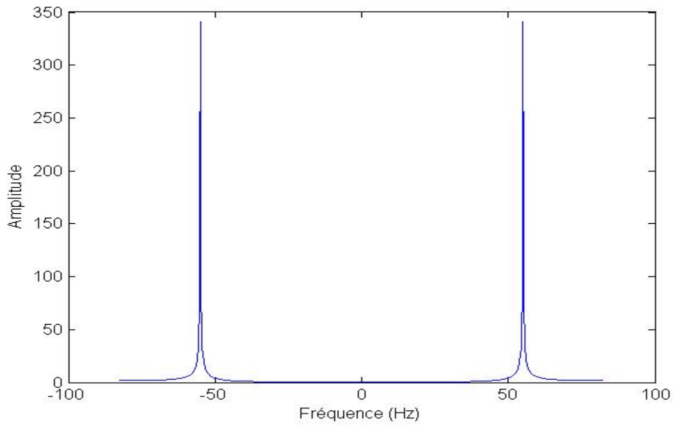

FIGURE 2.8 – Phénomène de repliement pour $f_{s_{rep}} = 1.5.f_0$

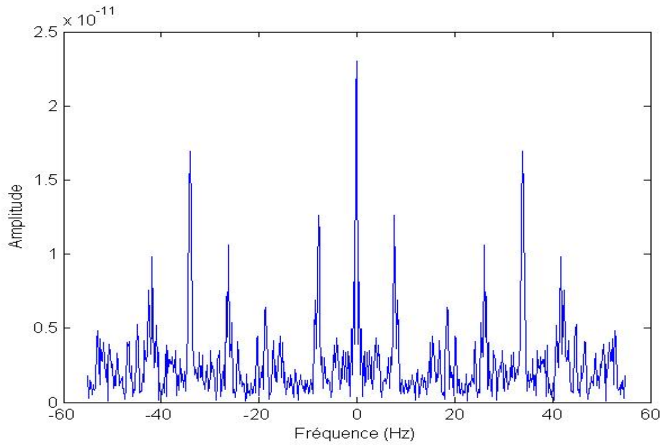

FIGURE 2.9 – Phénomène de repliement pour $f_{s_{rep}} = f_0$

Sur la FIGURE 2.8., la fréquence calculée par la FFT est fausse (en effet, le pic est à 55 Hz au lieu de 110 Hz) car une fois sur deux, la FFT retombe sur un point qui a la même valeur.
Sur la FIGURE 2.9., les points de l'échantillon sont choisis toutes les périodes, et ont donc tous la même amplitude, d'où la FFT représentant un signal continu. Avec plus de précision, on aurait juste un pic en 0.

2.4 Traitement simple d'un signal

On souhaite retrouver la fréquence initiale f_0 à partir de l'analyse fréquentielle du signal. Nous considérons donc les graphiques des transformées de Fourier (DFT ou FFT). Le signal n'ayant qu'une fondamentale, il n'y a qu'un pic qui indique la fréquence voulue. Il suffit donc de lire sur l'axe des abscisses la fréquence associée au pic (en valeur absolue...). Nous utilisions donc en MatLab la fonction `max` pour obtenir l'indice du pic et nous l'injectons dans le vecteur fréquence pour obtenir la fréquence du signal. Après exécution des lignes de code, nous trouvons bien 110 pour chaque méthode (DFT et FFT).
La deuxième étape est d'afficher une action à faire pour l'utilisateur si jamais la corde est désaccordée de moins d'un quart de ton. Pour cela, nous créons donc une fonction que nous appelons *consigne* dont le code est détaillé en Annexe A, paragraphe *Traitement simple d'un signal*. Pour considérer que la corde est accordée, on se laisse une marge d'erreur d'$\frac{1}{20}$ de Savart. Si la différence entre la fréquence du signal et l'objectif est plus important que la marge d'erreur, on indique à l'utilisateur qu'il doit Resserer (ou Desserer) sa corde.

2.5 Robustesse d'un traitement par rapport à un nouveau modèle de signal

Nous avons ajouté des harmoniques au signal sinusoïdal initial en leur donnant des amplitudes différentes, ceci dans le but de s'approcher d'un signal réel.

Pour la suite, la méthode de recherche de la fondamentale se déroule de la même façon. Bien entendu, la valeur affichée par notre code est fausse puisque nous avons volontairement elevé l'amplitude de l'harmonique de rang 3 (330 Hz) c'est donc cette harmonique qui est détectée. Ce problème peut se résoudre en utilisant un filtre passe-bande adapté à chaque corde.
On comprend alors la nécessité de chacune des cordes de n'être que modérément désacordée (un quart de ton). De cette manière, la fondamentale restera dans le passe-bande et son identification sera possible. Ci-dessous se trouve la FIGURE 2.10. représentant la DFT du nouveau signal :

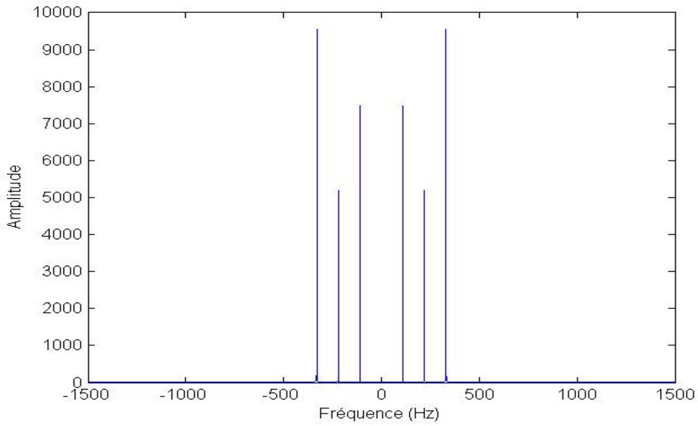

FIGURE 2.10 – Signal réel comportant 3 harmoniques d'amplitudes différentes

2.6 Question optionnelle

L'objectif de cette question est de permettre la lecture des fréquences des notes à partir d'une base de donnée sous la forme d'un fichier texte. Pour cela, nous avons utilisé les fonctions `textscan` et `fopen`. De plus, nous avons créer la fonction *reader* qui permet de réaliser cette opération.

Chapitre 3

Conclusion

Ce début de projet nous a permis de prendre en main le sujet ainsi que le logiciel MatLab. En effet, nous avons utilisé de nombreuses fonctions afin de créer des outils pour le traitement du signal qui nous serons utiles pour la deuxième partie.
Cette familiarisation avec le logiciel nous rend optimistes pour la conception de l'accordeur de guitare à proprement parler puisque le travail préliminaire du-dit accordeur est fait ; il ne nous reste donc plus qu'à concevoir et étudier des filtres numériques pour palier le problème des signaux réels que nous avons entrevu à la Question 5.

Annexe A

Codes MatLab

Cette annexe répertorie tous les codes que nous avons écrits pour ce projet. Les lignes de code sont davantage détaillées sur cette annexe que sur le M-file.

Construction d'un vecteur par concaténation

```
% Construction du vecteur e
  e = [9.5 7 5 2.5 0 -2.5];

% Construction de l'écart en fréquence
  % Construction d'un vecteur colonne rempli de zéros.
    f = zeros(1, length(e));
  % Vectorisation de la conversion de l'écart de ton en fréquence.
    f = 110*10.^(e*log10(2)/6);
    f;

% Longueur d'un demi-ton en Savarts
  e_dt = 1000*log10(2)/12;

% Longueur d'un ton en Savarts
  e_t = 1000*log10(2)/6;
```

Créer une fonction MatLab

Fonction *sav2lin.m*
Voici dans un premier temps le **script** que nous avons élaboré :

```
% Script pour la fonction sav2lin.m

  % Convertir les Savarts en échelle linéaire

    % Vecteur qui sera passé en argument de la fonction
      xsav = [0 25.08 50.17 301.03];
    % Construction d'un vecteur colonne rempli de zéro
      xlin = zeros(1, length(xsav));
    % Vectorisation de la conversion Savarts -> linéaire
      xlin = 10.^(xsav/1000);
      xlin;
```

Voici maintenant le **code source** du fichier *sav2lin.m* :

```
function [xlin] = sav2lin(xsav)

xlin = zeros(1, length(xsav));
xlin = 10.^(xsav/1000);

end
```

Fonction *lin2sav.m*
Voici dans un premier temps le **script** que nous avons élaboré :

```
% Script pour la fonction lin2sav.m

  % Convertir l'échelle linéaire en Savarts

    % Vecteur test qui sera passé en argument de la fonction
      xlin = [1 2 3];
    % Construction d'un vecteur colonne rempli de zéros
      xsav = zeros(1, length(xlin));
    % Vectorisation de la conversion linéaire -> Savarts
      xsav = 1000*log10(xlin);
      xsav;
```

Voici maintenant le **code source** du fichier *lin2sav.m* :

```
function [xsav] = lin2sav(xlin)

xsav = zeros(1, length(xlin));
xsav = 1000*log10(xlin);

end
```

Création et représentation du contenu d'un signal

Dans un premier temps, nous devons générer le signal sonore, ce que nous faisons en tapant le code suivant :

```
% Fréquence du signal
  f0 = 110;
% Temps durant lequel nous voulons jouer le signal
  T = 5;
% Fréquence d'échantillonage
  fs = 10*f0;
% Période d'échantillonage
  Te = 1/fs;
% Découpage de l'intervalle de temps à la période d'échantillonage
  t = [0:Te:T-Te];
% Signal sinusoïdale de fréquence f0
  signal = sin(2*pi*f0*t);
% Jouer le signal sonore
  sound(signal, fs);
```

Ensuite, nous voulons représenter le contenu du signal échantillonné dont voici le code :

```
% Définition de la graduation de l'axe des abscisses
  x=0:Te:10/f0;
% Définition de la graduation de l'axe des ordonnées
  y=sin(2*pi*f0*x);
% Représentation du signal
  figure;
  plot(x,y);
% Affectation de noms aux axes d'abscisses et d'ordonnées
  xlabel('Temps(s)');
  ylabel('Amplitude');
```

Fonction *DFT*

Implémentation du calcul de la DFT sur les 1000 premiers points du signal

```
  function [ S ] = DFT( y )

  l = length(y);
  N = ((0:1:l-1).')*(0:1:l-1);
  S = exp((-2*pi*1i*N)/l)*(y)';

  end
```

Représentation du contenu du signal sinusoïdal par un calcul de la transformée de Fourier discrète

```
% Calcul de la DFT du signal
  Sdft = abs(fftshift(DFT(signal)));
% Longueur du vecteur Sdft
  ldft = length(Sdft);
% Intervalle de fréquences de l'affichage
  fdft = -fs/2:fs/ldft:fs/2-fs/ldft;
% Représentation du contenu du signal
  figure;
  plot(fdft,Sdft);
% Définition des noms des axes
```

```
    xlabel('Fréquence(Hz)');
    ylabel('Amplitude');
```

Nous faisons la même chose en remplaçant notre fonction `DFT` par la fonction `fft` directement implémentée dans MatLab.

```
% Calcul de la FFT du signal
    Sfft = abs(fftshift(fft(signal)));
% Longueur du vecteur Sfft
    lfft = length(Sfft);
% Intervalle de fréquences de l'affichage
    ffft = -fs/2:fs/l:fs/2-fs/l;
% Représentation du contenu du signal
    figure;
    plot(ffft, Sfft);
% Définition des noms des axes
    xlabel('Fréquence(Hz)');
    ylabel('Amplitude');
```

Fonction *TransformeZ*

Implémentation du calcul de la transformée en Z

```
    function [ Sz ] = TransformeZ(f0, x, fbar)

    M = length(x);
    Sz = (1-exp(1i*2*pi*(f0-fbar)*M))/(1-exp(1i*2*pi*(f0-fbar)));

    end
```

Voici le code illustrant la précision de l'analyse par transformée de Fourier :

```
% Nouvelle fréquence fs
    fs = 300; % Hz
% Nouveau temps T
    T = 1; % sec
% Définition des points de la séquence
    m = 0:length(signal);
% Vectorisation de la séquence
    seq = exp(1i*2*pi*(f0/fs)*m);
% Définition de la séquence de points sur laquelle appliquer
% la transformée en Z
    fbar = linspace(0, 1, 3e3);
% Calcul de la transformée en Z
    Sz = TransformeZ(f0/fs, seq, fbar);
% Tracé sur une même figure de la DFT et de la transformée en Z
    figure;
    plot(fdft, Sdft);
    hold on;
    plot(fdft, Sz);
    xlabel('Fréquence(Hz)');
    ylabel('Amplitude');
```

Voici le code associé à la partie des représentations des différentes résolutions :

```
% Cas où deltaf = 1Hz
      deltaf = 1;
  % Vecteur temps
```

```
    t01 = 0:1/fs:T;
  % Signal sinusoidal initial
    yr1 = sin(2*pi*f0*t);
  % Signal sinusoidal décalé de deltaf
    yr2 = sin(2*pi*(f0-deltaf)*t);
  % DFT centré en 0 de la somme des deux signaux yr1 et yr2
    Yr = abs(fftshift(DFT(yr1+yr2)));

  % Représentation graphique
    figure;
    plot(fdft, Yr);
    xlabel('Frequence(Hz)');
    ylabel('Amplitude');

% Cas où deltaf = 0,5Hz
    deltaf = 0,5;
    t01 = 0:1/fs:T;
    yr1 = sin(2*pi*f0*t);
    yr2 = sin(2*pi*(f0-deltaf)*t);
    Yr = abs(fftshift(DFT(yr1+yr2)));

    figure;
    plot(fdft, Yr);
    xlabel('Frequence(Hz)');
    ylabel('Amplitude');

% Cas où deltaf = 0,2Hz
    deltaf = 0,2;
    t01 = 0:1/fs:T;
    yr1 = sin(2*pi*f0*t);
    yr2 = sin(2*pi*(f0-deltaf)*t);
    Yr = abs(fftshift(DFT(yr1+yr2)));

    figure;
    plot(fdft, Yr);
    xlabel('Frequence(Hz)');
    ylabel('Amplitude');
```

Nous donnons enfin le code permettant d'illustrer le phénomène de repliement :

```
% Fréquence d'échantillonage égale à 1,5 fois la fréquence du signal
    fs_rep1 = 1.5*f0;
  % Vecteur temps
    t_rep1 = 0:1/fs_rep1:5;
  % Signal sinusoidal echantillonné
    y_rep1 = sin(2*pi*f0*t_rep1);
  % DFT du signal y_rep1
    Y_rep1 = abs(fftshift(DFT(y_rep1)));
  % Longueur de Y_rep1
    l_rep1 = length(Y_rep1);
  % Intervalle de fréquences de l'affichage
    fdft_rep1 = -fs_rep1/2:fs_rep1/l_rep1:fs_rep1/2-fs_rep1/l_rep1;

  % Représentation de la DFT
    figure;
    plot(fdft_rep1, Y_rep1);
    xlabel('Fréquence (Hz)');
```

```
    ylabel('Amplitude');

% Fréquence d'échantillonage égale à la fréquence du signal
    fs_rep2 = f0;
    t_rep2 = 0:1/fs_rep2:5;
    y_rep2 = sin(2*pi*f0*t_rep2);

    Y_rep2 = abs(fftshift(DFT(y_rep2)));
    l_rep2 = length(Y_rep2);
    fdft_rep2 = -fs_rep2/2:fs_rep2/l_rep2:fs_rep2/2-fs_rep2/l_rep2;

    figure;
    plot(fdft_rep2, Y_rep2);
    xlabel('Fréquence (Hz)');
    ylabel('Amplitude');
```

Traitement simple d'un signal

Nous voulons retrouver la fréquence initiale f_0 à partir de l'analyse fréquentielle du signal, pour cela, nous entrons ces lignes de commande dans le script MatLab :

```
% En utilisant la DFT
  % Indice de la valeur maximale de Sdft
    [m1, ind1] = max(Sdft);
  % On trouve f0
    f_initiale1 = abs(fdft(ind1));

% En utilisant la FFT
    [m2 ,ind2] = max(Sfft);
    f_initiale2 = abs(ffft(ind2));
```

Fonction *consigne*
Afficher pour l'utilisateur l'action à faire sur sa corde.

```
function [ ] = consigne( freq, f0 )

  ton = 100/6*log10(2);                 % un ton
  precision = ton/20;                   % Marge d'erreur tolérée
  delta = lin2sav(freq/f0);             % Différence entre la corde
                                        % et l'objectif (en Savart)

  if(delta > precision)
      fprintf('Desserer \n')            % Affichage
  else
      if(delta < -precision)
          fprintf('Resserer \n')        % Affichage
      else
          fprintf('Corde accordée \n')  % Affichage
      end
  end

end
```

Voici enfin les instructions que nous avons mises dans notre script pour utiliser la fonction *consigne*, elle permet d'afficher la consigne pour accorder la guitare.

```
% On remplace 109 par la fréquence de la corde que l'on veut accorder.
  frequence = 109;    %% A MODIFIER %%
% Appel de la fonction consigne
  consigne(frequence, f0);
```

Robustesse d'un traitement par rapport à un nouveau modèle de signal

Ci-dessous le code MatLab relatif à l'analyse du nouveau signal comportant 3 harmoniques :

```
% Ajout des harmoniques
  fs = 10*fs;
  T = 5;
  t = 0:1/fs:T;
  l = length(t);
% Intervalle de fréquences pour l'affichage
  f = -fs/2:fs/l:fs/2-fs/l;
% Nouveau signal avec 3 harmoniques
  y = sin(2*pi*f0*t) + 0.7*sin(4*pi*f0*t) + 1.3*sin(6*pi*f0*t);

% Analyse par dft
  S = abs(fftshift(DFT(y)));

  figure;
  plot(f, S);
  xlabel('Fréquence (Hz)');
  ylabel('Amplitude');

% Recherche de f0
  [m, ind] = max(S);
  f_initiale = abs(f(ind));

% Consigne d'accordage
  consigne(f_origine, f0);
```

Question optionnelle

Fonction *reader*
Fonction permettant de scanner un fichier texte pour en retirer les fréquences des harmoniques contenues dans le fichier texte.

```
function [ frequence ] = reader( note )

fileID = fopen('freq_gamme.txt', 'r');
C = textscan(fileID, '%s %s');
l1 = length(note);

    for i = 1:35
        if(length(C{1}{i}) == l1)
            if (C{1}{i}(1:length(C{1}{i})) == note(1:l1))
                frequence = C{2}{i};
            end
        end
    end
fclose(fileID);

end
```

Enfin, voici les instructions que nous avons écrites dans le script pour faire appel à cette fonction.

```
reader('MI3');
reader('MI2');
reader('MI1');
```