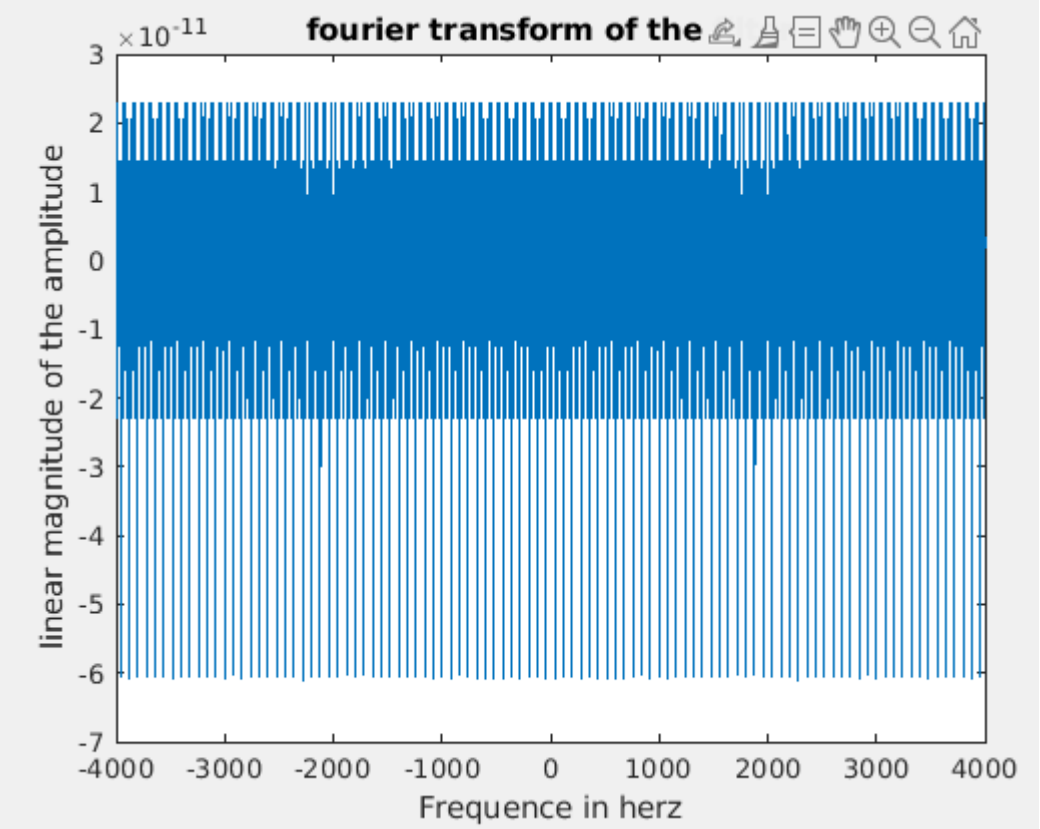
Mouhameth Yaté
Israel B Kafando

Compte rendu Projet Processeurs de Traitement de signal

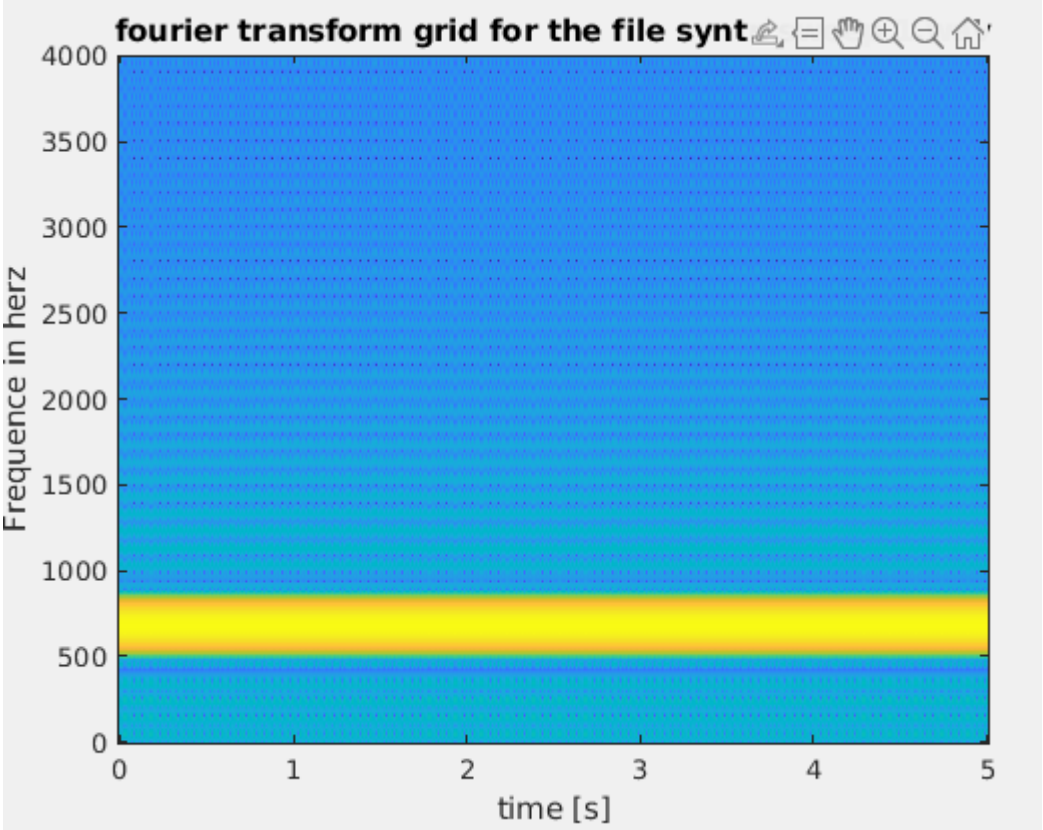
.1 First iteration: Signal analysis, basic compilation .
3.1.1 Analysis part
Dans cette première partie, notre démarche consiste à examiner et analyser divers signaux audio. Pour mener à bien nos tests, nous utiliserons les fichiers audio du dossier Base_Sound.

1. Load a file and check the properties of the signal. Plot the signal in time and frequency domain. You can also listen to the audio file (with headphones for the sake of your professor). For each signal, be sure you be able to describe the signal in time and frequency domains

On analyse les signaux avec la fonction Matlab audio_analysis qui prend en entrée un fichier, qui donne le signal **sig** et la fréquence à laquelle il a été échantillonné (**fs)**. On a différents signaux audio , on fera l'analyse temporelle et fréquentielle sur quelques un de ces signaux.
L'analyse du signal en temporelle n'est pas pertinente pour les signaux audios. On passe donc dans le domaine dual (domaine fréquentielle) pour faire une analyse beaucoup plus fine.
synth_beep_1 :

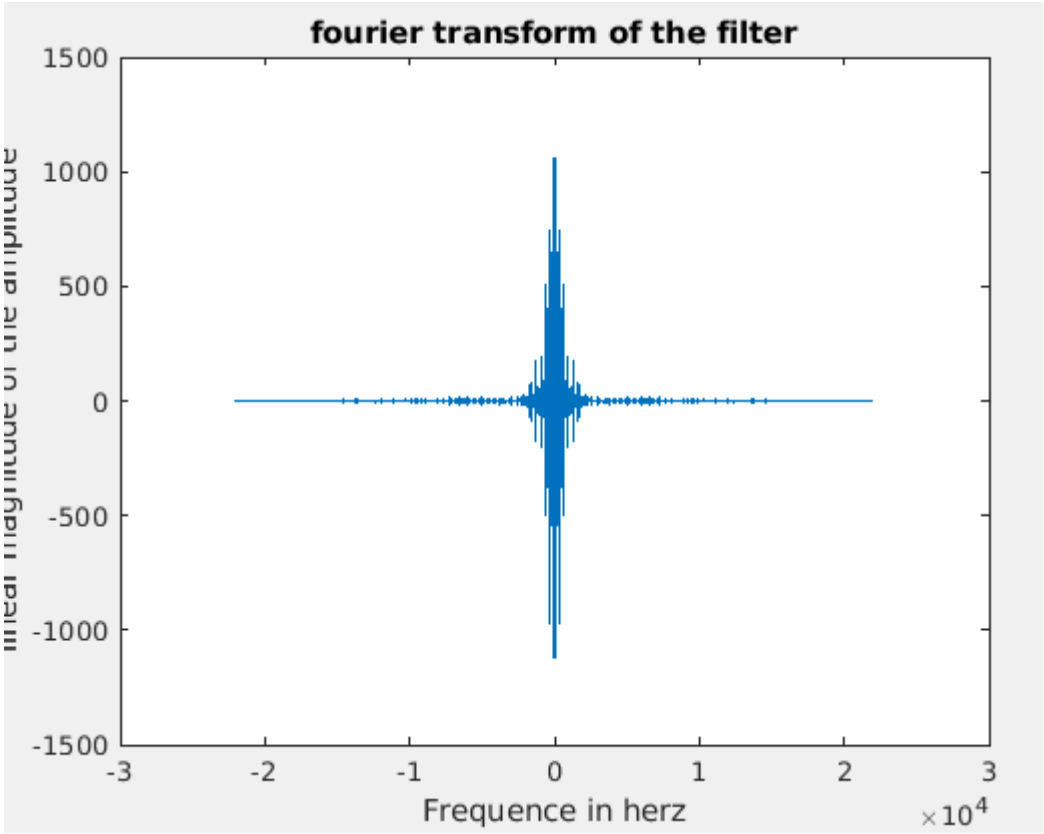


Dans le domaine fréquentiel on voit que l'amplitude du signal est constante sur toute les plages de fréquence. C'est pourquoi on entend un son avec une tonalité constante.

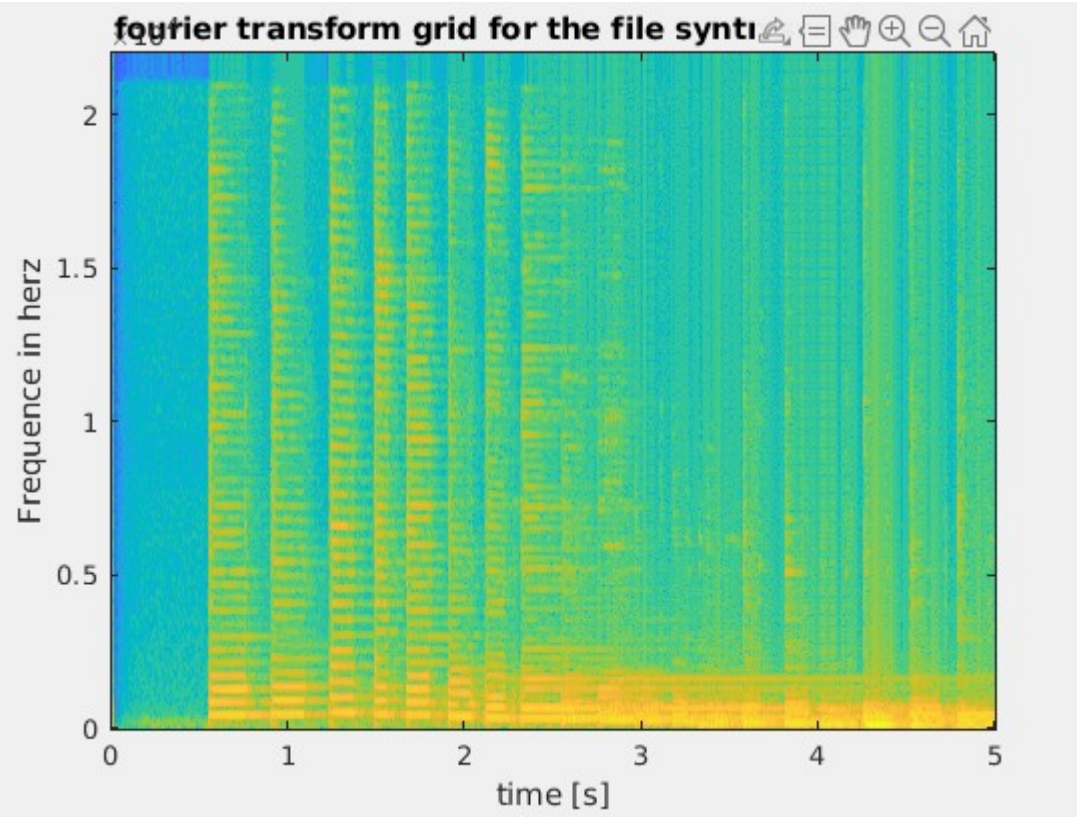


La représentation temps-fréquence confirme le fait qu’on a une fréquence constante sur toute la durée du signal audio.

guitar_1 :

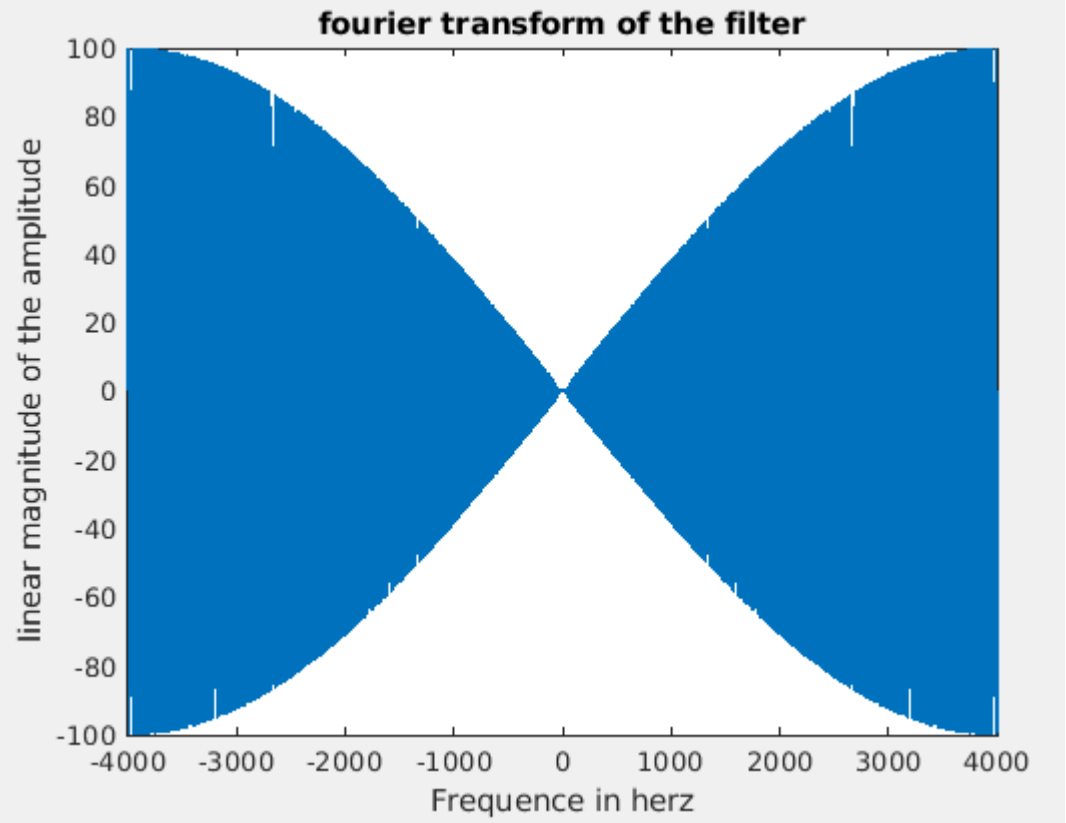


En fréquentielle, on note une raie principale avec des harmoniques pour guitar_1. On s'y attendait car la guitare produit un son qui est une superposition de notes , et les notes sont des harmoniques de fréquence différentes.

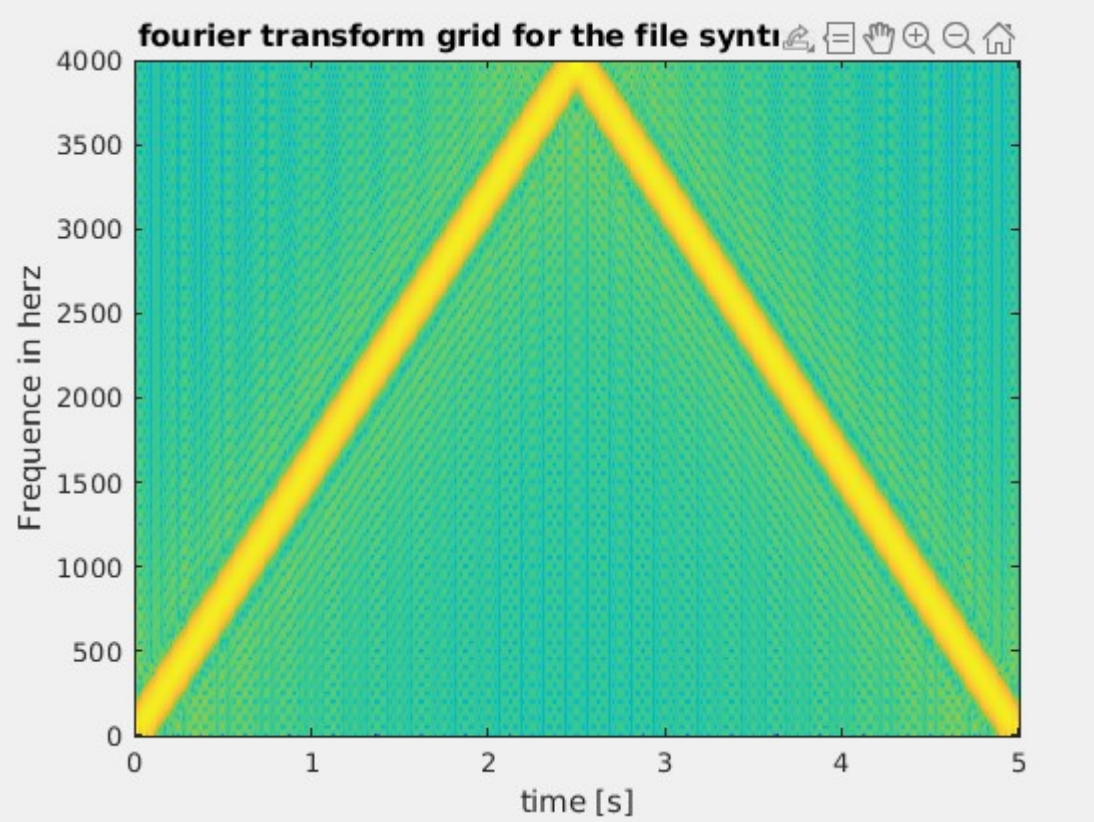


On remarque qu'en temps-fréquence que les harmoniques se superposent. En effet pour une fréquence donnée on observe qu'il est présent sur toute la durée du signal. C'est pourquoi on a la superposition des harmoniques.

synth_sweep_1 :



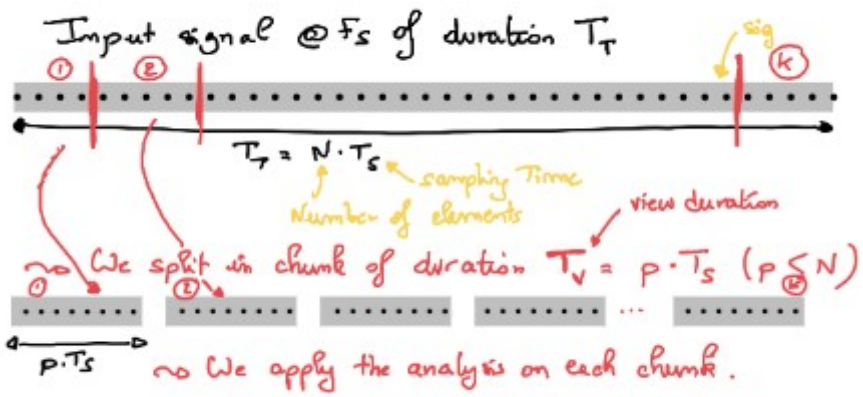
Dans le domaine fréquentiel on voit que l'amplitude du signal augmente suivant la fréquence jusqu'à atteindre une valeur maximale puis diminue jusqu'à s'annuler.
On a représenté le signal dans le domaine fréquentiel en prenant en compte les fréquences négatives. La partie négative n'est rien d'autre que la TF du signal entre fs/2 et fs. Ce qui est exactement le comportement en fréquentiel qu'on avait prévu.



Dans la représentation temps fréquence, on voit que la fréquence augmente au cours du temps de façon linéaire jusqu'à atteindre une valeur maximale puis diminue. Ceci explique par le fait qu'on a l'effet Doppler quand on écoute le signal synth_sweep_1.

2a)Chunk the input signal into different pieces, each of them with desired duration (that we will call a view). Note that this duration may not be aligned with the FFT size. This duration should be define in seconds and corresponds to a few millisecond of signal. Note that the few samples of the end can be discarded if it does not fit in a view.

Maintenant on essaye de visualiser le signal en temps-fréquence. Il s'agit d'analyser la composante fréquentielle dans une fenêtre temporelle. Cette fenêtre est un paramètre que l'on choisira en rapport avec la nature du signal. On a un signal audio échantillonné à la fréquence Fs, donc une durée d'observation d'au moins Ts égale à 1/Fs est suffisant . Une fois que Tt est connue, on décompose notre signal en petit vecteur de taille p.Ts. À partir de ces vecteurs on construit k vecteurs de taille p.Ts.

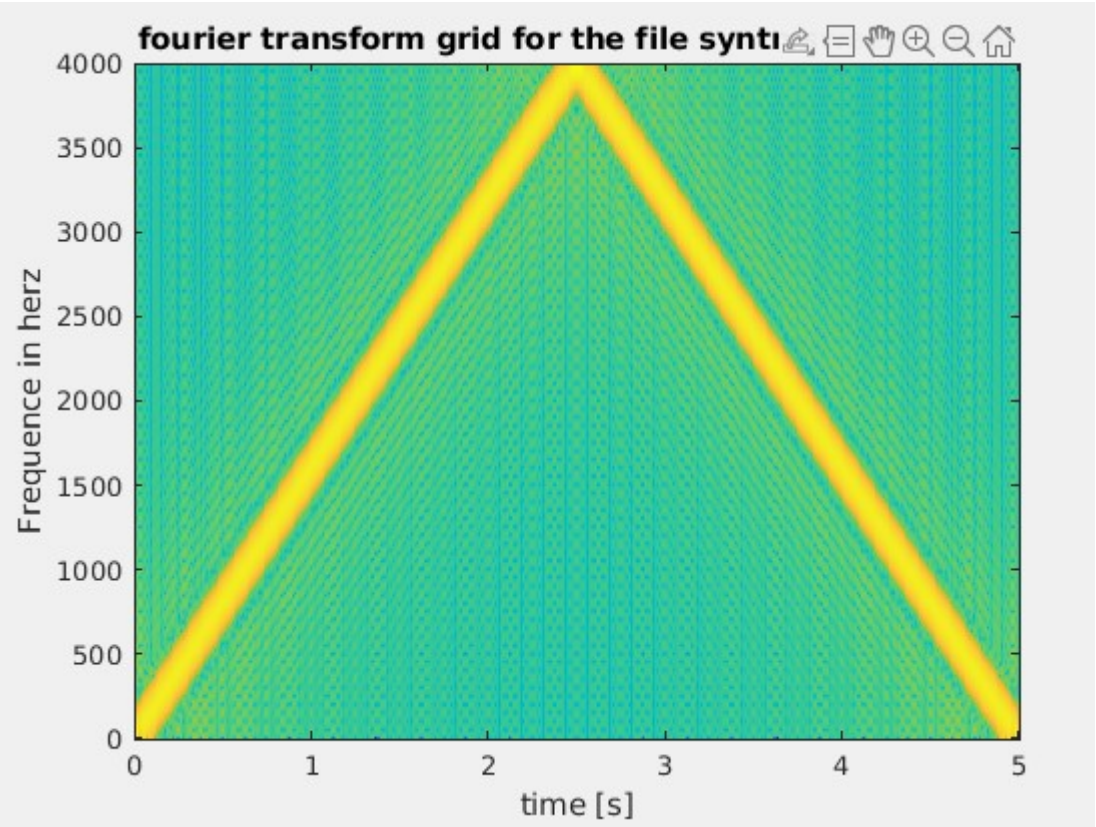


b) for each view, apply an apodization function. The apodization function should be chosen carefully, see the associated DSP lecture part.
On a choisi d'appliquer la fenêtre de hamming.

(c) Apply the frequency transform to each view. If the view size is lower than the FFT size you should follow a Zero Padding approach. You can extract the modulus on one side and the angle on another matrix.

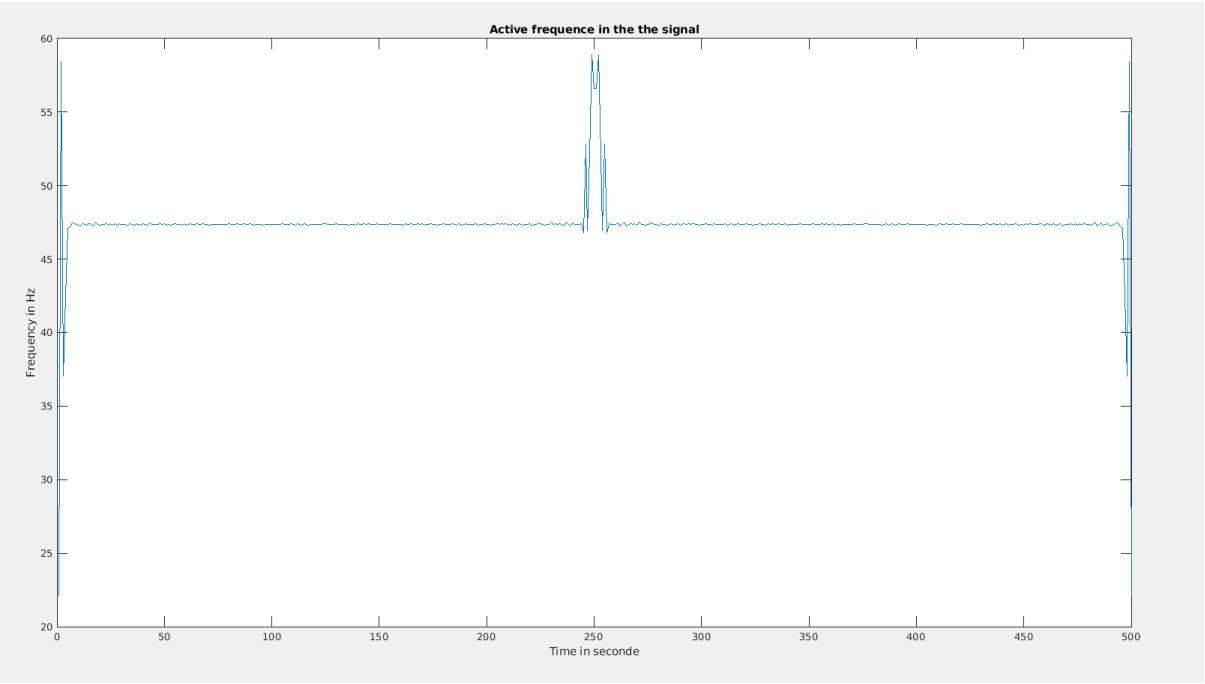
Voir annexe 1, pour le code MAtlab de la fonction time_frequency qui prend en entré un signal x , la fréquence à laquelle le signal a été échantillonnée **fs** , la taille du fft **taille_fft** et enfin **d** qui es la durée de la fenêtre d'observation. On a suivi l'approche du zéro padding car la taille des vecteurs n'étaient pas une puissance de 2. On a donc complété les vecteurs avec des zéros afin qu'ils fassent la taille fft souhaitée en précisant dans la fonction matlab la taille (voir code matlab). Ces derniers sont stockés colonne par colonne dans une matrice carré de taille taille_fft. On prend le module et la phase de cette matrice.

(d) You have now two matrixes as you have frequency components (modulus and phase) for each view. Display the matrix (using imagesc) of the modulus (as it is in Figure 3.1). By default, imagesc flips the axis and you can use the Matlab command set(gca,'YDir','normal');
La matrice obtenue est donc une représentation de la fréquence en fonction du temps car la durée **d** sépare deux colonnes successives et que la fréquence croît d'une ligne à une autre. On affiche maintenant la matrice avec la fonction image de matlab on obtient la figure



3. Perform the same kind of analysis for the other signal. What is the active frequency used in synth_beep_1 ? What is the active frequency used in synth_beep_2 ? For this two algorithms, propose a algorithm that automatically extract the main frequency component (a pitch) and checks that the result you have are coherent with the T/F grid you have ploed.

On a créé une autre fonction appelée active_fréquence qui prend en entrée la matrice temps-fréquence et qui relève la fréquence maximale sur chaque colonne. On récupère avec la fonction max de matlab la fréquence maximale de chaque colonne. Le résultat est sur la figure suivante.



3.1.2 Implementation part

Write a function populate_dma(ptr0,sig,size) that copies an array sig at the location where it can be used by the I2S interface. We indeed work at another location and then copy the location of interest at the DMA location. An illustration of the memory mapping is presented on Figure 3.2. Two (very) important remarks

Les fichiers audios sont codés sur 16 bits mais vu qu' on veut écouter le signal audio sur deux écouteurs , on les cast sur 32 bits pour dupliquer le signal audio. Le signal audio sera stocké en mémoire, on utilisera donc un pointeur **ptr0** pour récupérer au fur et à mesure le signal dans l'espace mémoire. On contrôle la récupération du signal avec sa taille pour éviter de taper dans une zone mémoire non autorisée.

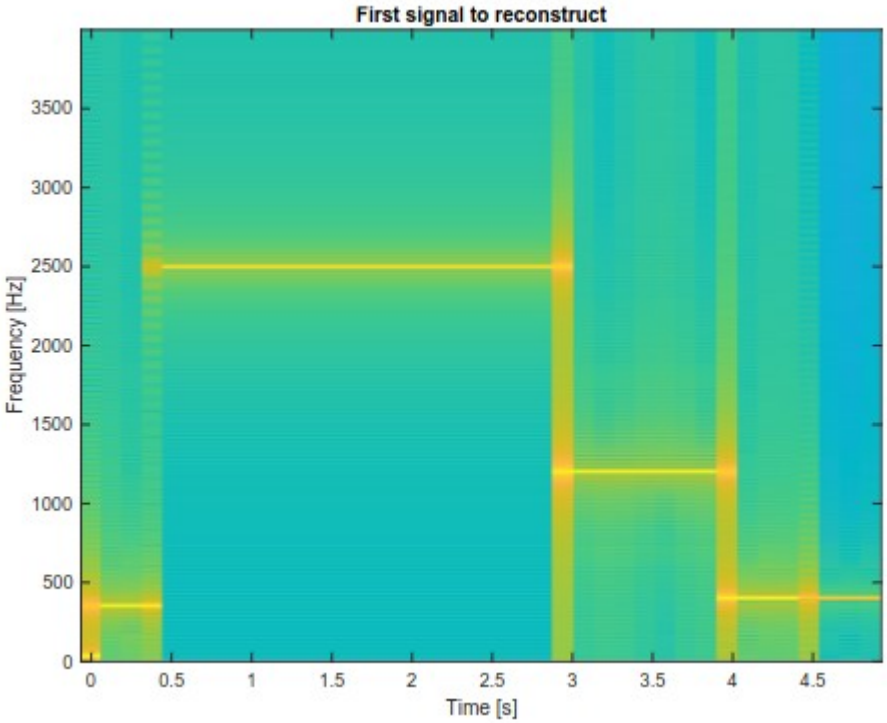
3.2 Second iteration: Pitch synthesis

3.2.1 Analysis part

1)Propose an algorithm that is able to construct a similar signal as the one proposed in Figure 3.3. You have to first analyze the image to extract the core characteristics of the

signal and then inject these characteristics into a script generate_synth_sig.m. You can also load the Matlab figure to beer zoom if you want. Remember that the image corresponds to a T/F grid and the associated sine wave has the form si n(2π f t) where f is the sine frequency and t the time axis.

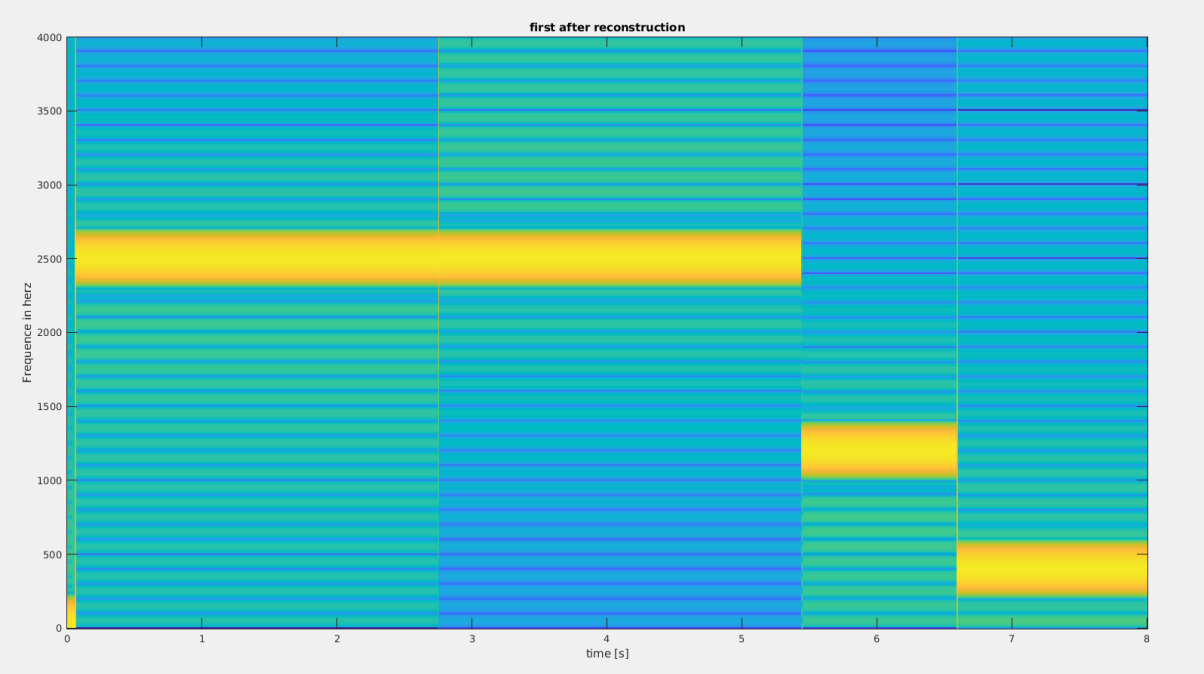
Dans cette partie on va reconstruire un signal en se basant sur sa représentation T/F. Le signal qu'on doit reconstruire est dans la figure suivante :



On relève les fréquences avec leur temps d'apparition dans le tableau ci-dessous.

Temps [s]	0 - 0.064	0 - 0.448	0.32 - 3.008	2.88 - 4.032	3.504 - 4.915
fréquence(Hz)	30	350	2500	1200	400

On écrit la fonction reconstruct_signal.m qui concatène ces sinus de fréquence Fe à 8 kHz avec leur intervalles respectifs pour obtenir notre signal complet reconstruit. Le signal ainsi obtenu est mis en entrée de la fonction time_frequence pour afficher la représentation T/F. On obtient le résultat de la figure suivante :



On voit bien que le signal obtenu n'est pas exactement le même que le signal à reconstruire. En effet, cela est dû au fait que l'on ne peut pas décrire parfaitement un signal dans les deux domaines duaux en même temps. Plus on est précis en temporelle , moins on le sera en fréquentiel et inversement.

3. As you have done in Matlab, the signal generation is done with sine wave. On the Zybo board all operations are in Fixed Point and the sine wave should also follows this formalism. To do so we will have to find an estimation of a sine wave that allows us to synthetize kind of a sine wave. We will call this function fixSine. This function will take a Int16 as an input and produce an Int16.

(a) We will use a Taylor approximation for the sine function. Trough Taylor expansion we canquickly find the expression of an approximation of the sine function defined by for $x \in]-\pi/2; \pi/2[$. We will use an intermediate variable $\underline{x}$ and define the approximation as

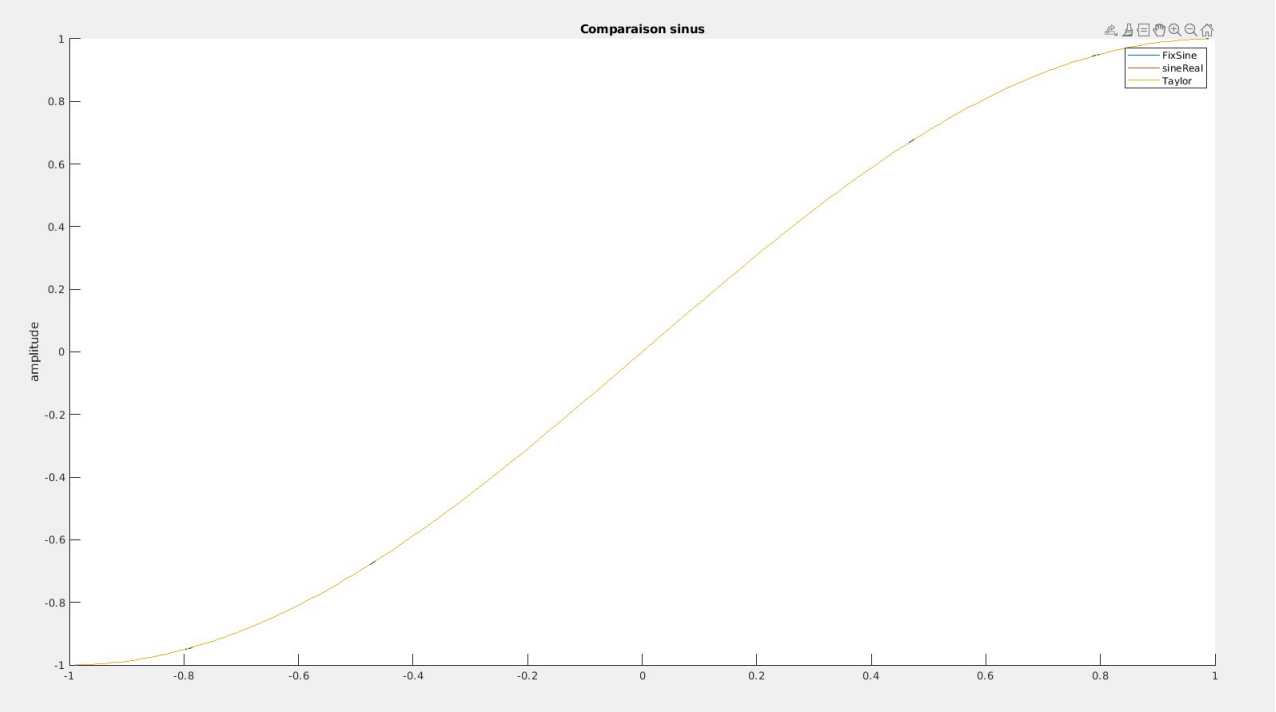
$$\underline{P(x)} = a.\underline{x} + b.\underline{x^3} + c.\underline{x^5}$$

Les questions 3)a , 3)b , 3)c et 3)d sont traitées sur l'annexe 1.

e.) Implement a fixed point function in Matlab. This function should take an int16 as input and returns an int16. You can use the function provided that performs multiplication and shifts as it will be on the Zybo.

La fonction fixSine prend en entré un entier (en virgule fixe) codé sur 16 bits sous le Q(16,0,15) et retourne un entier aussi codé sur 16 bits sous le Q(16,0,15) car il s'agit d'un sinus . Pour avoir une meilleure comparaison , on superpose le sinus réel , avec le sinus approché en 3)a (approximation polynomiale) et le sinus obtenu en fixe code. On obtient la

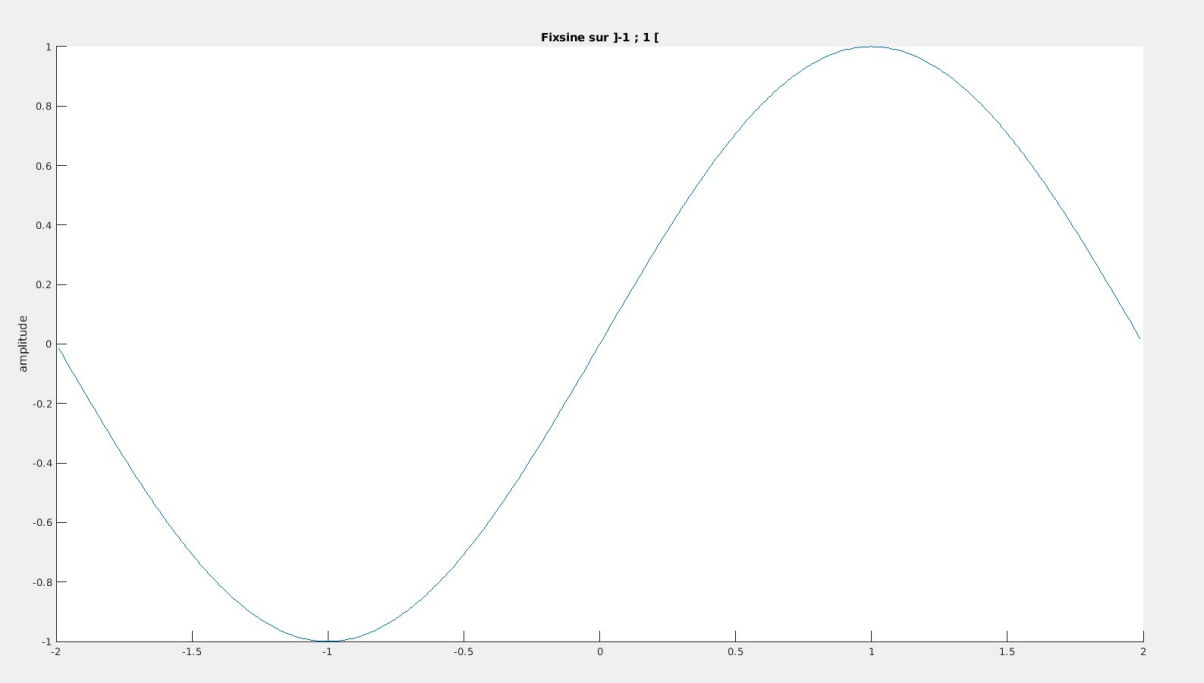
figure suivante lorsqu'on superpose les trois sinus entre et]-1 ; 1[.



Les trois représentations du sinus se superposent parfaitement sur]-1 ; 1[..

f. The Matlab function is capable to generate a sine for an angle between −π/2 and π/2. It will not be enough to be able to synthesize a complete sine wave. Using the symmetry properties of sinus, propose a function si n f pv 2 in Matlab that still takes an Int16 but in another format (to be defined) such that we can approximate the sinus for an x between] − π; π[. This is a domain extension that will allow us to generate sine waves more easily.

En codage sur 16 bits , l'intervalle]-1 ; 1[. devient un ensemble d'entiers compris entre -32768 et +32767. Et donc −π/2 , 0, −π/2 prennent respectivement les valeurs -16384 , 0 et 16384. En tenant compte de la symétrie su sinus on a coder sinfpv2 avec trois conditions :
x entre]−π/2 , π/2 [
x entre]π/2 , π [
x entre]−π , - π/2 [
Voir annexe et code matlab sinfpv2.



g. Explain why you will need to calculate ω = 4F c/F s on a Q(16,1,14) format. How can modular arithmetic can help you to ensure the result will still be in the sine definition domain ? When prototyping on Matlab, use the provided function add_int16_int16 that emulates the modular arithmetic of the Zybo adder. Write a Matlab function that generates a sine wave for a given frequency and check that the audio rendering is satisfying.

Pour calculer une sinusoïde à la fréquence f, on calcule le *sin(2* π*f*t)* et pour un signal échantillonné à une fréquence fs on a les points de la sinusoïde en *sin(2* π*(f/fs)*n)* = *sin(w*n) avec w = 2* π*(f/fs)*. Notre fonction sinfpv2 utilise w = (2*w)/π *ce qui donne finalement w = (2*2* π*(f/fs))/π = 4*(f/fs) qui sera sur un format de Q(16, 1, 14) en raison du théorème de Shannon qui impose une fréquence d'échantillonnage fs d'au moins deux fois plus supérieur à la fréquence (maximale) f du signal. Pour fs > 2f on a w < 2 qui tient bien sur un format de Q(16, 1, 14).*
Étant donnée que les additions sont réalisées sur des entiers de 16 bits on constate qu'il y aura un débordement quand la valeur de la somme dépassera 32767 et recommencera à -32768. Ces valeurs extrêmes sont bien dans la plage]-2, 2[(en virgule fixe) que peuvent supporter notre fonction sinfpv2.

3.2.2 Implementation part
3.3 Third Iteration: Pitch detection and Resynthesis
3.3.1 Analysis part
3.3.2 Implementation part
3.4 Fourth iteration: Autotune
3.4.1 Analysis part
3.4.2 Synthesis part