AMATH 482 Homework 2

Rui Zhu

Feb 4, 2021

Abstract

This project will analyze a portion of two of the greatest rock and roll songs of all time: *Sweet Child O' Mine* by Guns N' Roses and *Comfortably Numb* by Pink Floyd. We will reproduce the music score for the guitar in the GNR clip, and the bass in the Floyd clip. Since there are more than one musical instruments in the Floyd clip, we will use a filter in frequency space to try to isolate the bass and guitar solos in Comfortably Numb.

1 Introduction and Overview

Fourier Transform (FT) takes a function of space (or time), x, and converts it to a function of frequencies, k. However, Fourier Transform assumes infinite domain so is not necessarily what we want for practical situations. In last assignment, we used Fast Fourier Transform (FFT) to locate a submarine by finding the frequency centre. However, during that process we lost all the time information. Thus in this project, we will use Gabor Transform instead, to obtain a balance between time and frequency.

With the Gabor transform we can recover information about a signal in both the time and frequency domain. This was achieved by introducing the two key principles for joint time-frequency analysis: translation of a short time window and scaling of the short-time window to capture finer time resolution. The shortcoming of this method is that it trades off accuracy in time for accuracy in frequency and vice-versa. Thus, we will try different parameters before we really dive into the audio data.

2 Theoretical Background

2.1 The Gabor Transform

Briefly speaking, the implementation of Gabor Transform is fairly straight- forward - we just multiple the signal by a (discrete) window function (or filter) and then apply the FFT. We then move the window and repeat the process. The main reason we adopt a Gabor Transform here is that a (Fast) Fourier Transform resolved the frequency domain so 'well' that we lost all the information about time, take the GNR audio clip for example:

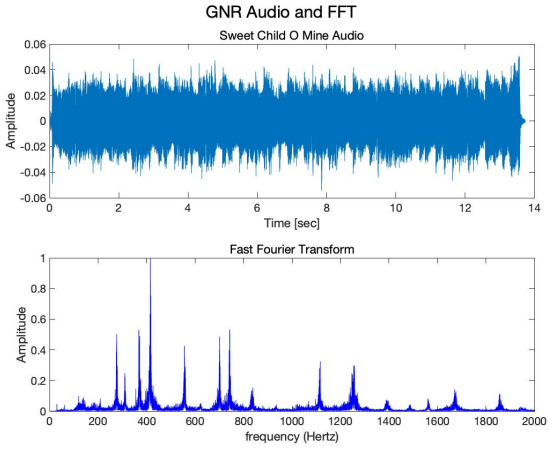

Figure 1: GNR audio and FFT

Thus, we need Gabor Transform which can maintain accuracy in both the time and frequency domains simultaneously, the only problem is that we need to carefully determine our filter so that we obtain an acceptable trade-off between time and frequency.

Let us consider a filter function $g(t)$. Then, shifting by τ and multiplying by a function $f(t)$ represents the filtered function $f(t)g(t-\tau)$, with the filter centred at τ. The Gabor transform, or the short-time Fourier transform (STFT), is then given precisely by:

$$\tilde{f}_g(\tau, k) = \int_{-\infty}^{\infty} f(t)g(t-\tau)e^{-ikt}\,dt \tag{1}$$

Notice that there are many choices for g and some commonly used assumptions are:

1. The function g is real and symmetric.

2. The L_2-norm of g is set to unity

In this project we will mainly specify our $g(t)$ to be a Gaussian function:

$$g(t-\tau) = e^{-a(t-\tau)^2} \tag{2}$$

where a and τ represent the width of the window and the centre of the window, respectively. Notice that small a means a wide window and large a means a thin window. In this project we will try different values of a and τ for the best resolution.

3 Algorithm Implementation and Development

3.1 Initial Setup

1. Read the audio data
2. Obtain the Fourier Modes n and record time in seconds (time domain L)
3. Define our domain and grid vectors
    - Notice that we rescale our k by $\frac{1}{L}$ instead of $\frac{2\pi}{L}$ because the unit of frequency we've been using: MATLAB scales out the 2π in the periods of oscillation to make the periods integers.
    - We also need to use `fftshift` to shift the transformed signial back to the 'normal' order.
4. Perform the normal FFT and observe zero information in time as we expected before.

3.2 Testing different filters for our audio clips

1. Test different value of a and τ of our Gabor fliter to make sure we have an acceptable trade-off between time and frequency. We also want to avoid over-sampling and under-sampling issues: that is, if our filters are too crowded (overlaping), we are assigning unnecessary computational load to our computers without gaining much new information; On the other hand, if our filters are too sparse then there will be clips of our audio that are not sampled at all, which will lead to poor spectrogram resolution.

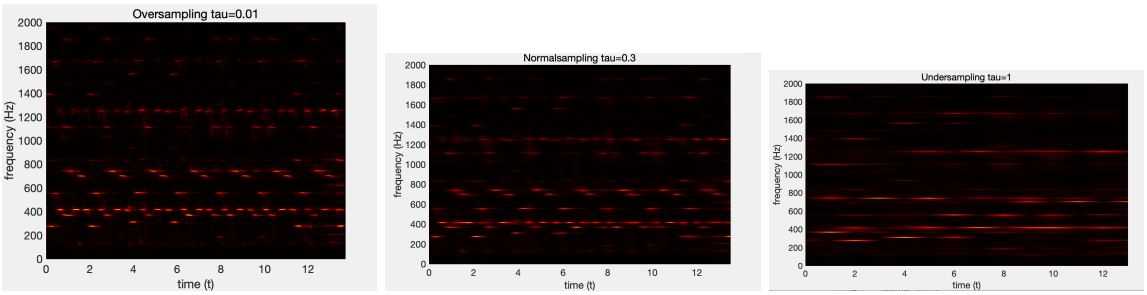

Figure 2: Over/Normal/Under-sampling

Algorithm 1:

Data: `GNR.m4a`

for *j in vector a (values of window width we want to test out)* **do**
  Define our spectrogram matrix which will be cleared at each loop iteration
  for *jj in τ vector (values of center we want to test out)* **do**
    Define our Gaussian filter function g
    Multiply the audio signal by the translating window function, then take the Fourier transform of the resulting function
    Take the absolute and use `fftshift` of the current result and save it as a new row of our spectrogram matrix
  Plot the spectrogram with current a-value with `pcolor` where x-axis represents the time and y-axis represents the frequency(Hz)

2. By observing and comparing the spectrograms we got, we chose our a: window width to be 100 and *tau*: center to be 0.3

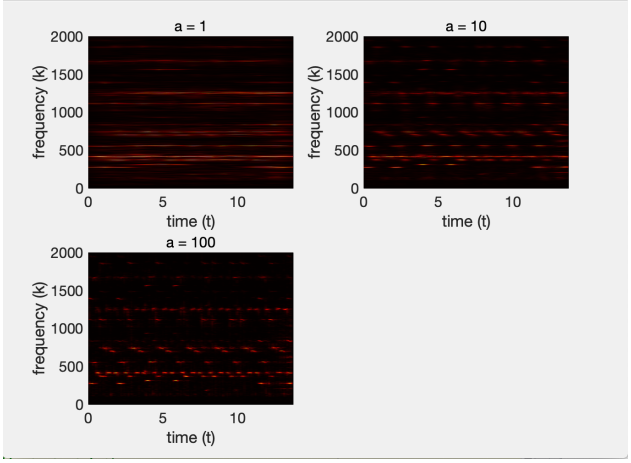

Figure 3: Testing different Window Sizes

3.3 Reproduce the music score

The algorithms of finding the music score are generally very similar to what we have done in previous section: Testing different filters for our audio clips. Expect that we:

- Loop only once through the whole audio clip with our choice of τ $(1 : \tau : L)$
- Find the center frequencies (place of largest magnitude) using `max() and abs()` and store them as the music scores at each specific time

On the other hand, algorithms for the `Floyd.m4a` clip are also quite similar, expect that we:

- Apply a Gaussian filter in frequency domain to focus on the the bass only (60-250Hz):

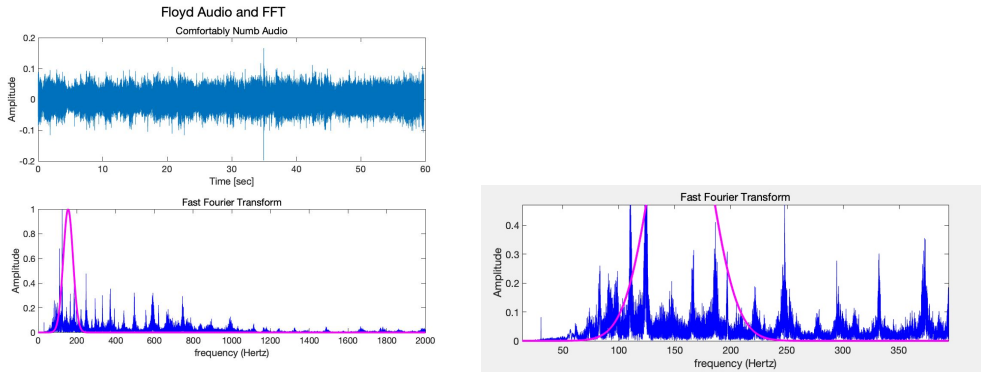

Figure 4: Isolating Bass in Floyd Clip Using Gaussian Filter

4 Computational Results

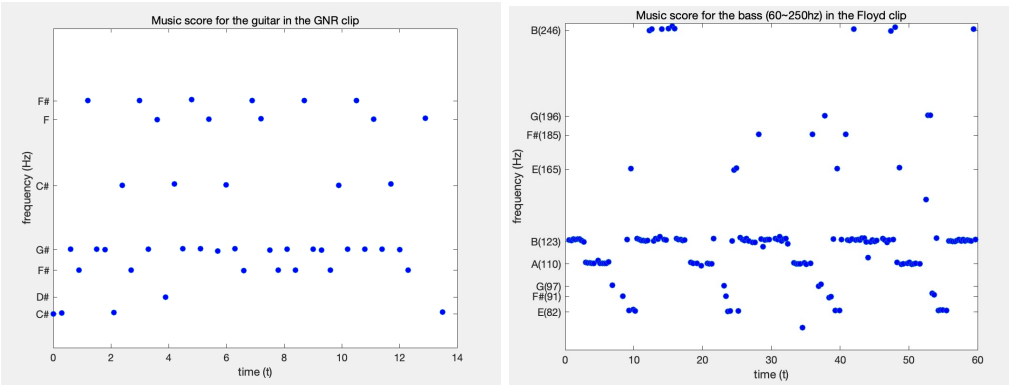

Figure 5: Music Score for GNR and Floyd Clip

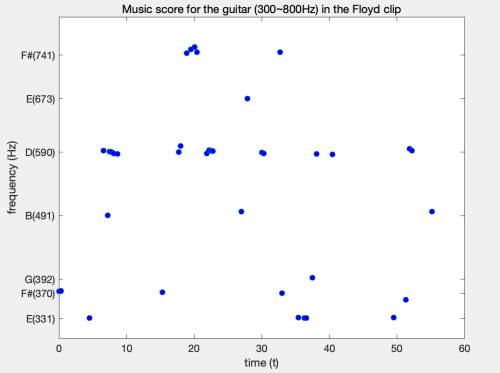

Figure 6: Music Score for Guitar Solo in Floyd Clip

5 Summary and Conclusions

Two audio clips were analyzed using time-frequency analysis. The Gabor transform was successfully implemented to create spectrograms and music scores of these two signals. A Gaussian filter was used as our Gabor filter. Different window widths and centres were tested and carefully picked. Undersampling and Oversampling were also simulated. Based on the music scores, our algorithm for GNR performed pretty well, and algorithm for Floyd performed slightly less satisfying (notice those little squiggles in the Floyd's music score). Again, this was caused by the trade-off between time and frequency, but the overall result is acceptable.

Appendix A MATLAB Functions

- `length(y)` rows and `length(x)` columns.
- `fftn()` decomposes the signal into frequencies in each direction.
- `ifftn()` decomposes the signal into time/space in each direction.
- `fftshift()` the transformed ut has an order of $k = 0, 1, ..., \frac{N}{2} - 1, -\frac{N}{2}, ..., -1$ this function shift it back to match the ut with frequency
- `audioread()` [y, Fs] = audioread('.m4a') where y is the audio data in the file, returned as an m*n matrix, where m is the number of audio samples read and n is the number of audio channels in the file, and Fs is the Sample rate, in hertz, of audio data y, returned as a positive scalar
- `yticks` sets the y-axis tick values, which are the locations along the y-axis where the tick marks appear. Specify ticks as a vector of increasing values; for example, [0 2 4 6]

- `yticklabels` sets the y-axis tick labels for the current axes. Specify labels as a string array or a cell array of character vectors; for example, {'Jan','Feb','Mar'}
- `pcolor(tau,k,spec)` draws a pseudocolor plot, we use this command to plot our spectrogram

Appendix B MATLAB Code

```
%% Data Import & Initial Setup


% Clean workspace
clear all; close all; clc


figure(1)
[y, Fs] = audioread('GNR.m4a');
% y:  audio data in the file, returned as an m*n matrix, where m is the
% number of audio samples read and n is the number of audio channels in
% the file. Here y is a single column vector, we will use the transpose of
% y to avoid potential problem when applying Gabor filters later
% Fs:  Sample rate, in hertz, of audio data y, returned as a positive scalar.


Y = y';
Yt = fft(Y);
n = length(Y); % Fourier modes
L = n/Fs; % record time in seconds (time domain L)
t2 = linspace(0,L,n+1);
t = t2(1:n);
k = (1/L)*[0:(n/2 - 1) -n/2:-1]; % Notice the 1/L instead of 2*pi/L
ks = fftshift(k);


% Plot audio
subplot(2,1,1);
plot(t,Y);
xlabel('Time [sec]');
ylabel('Amplitude');
```

```
title('Sweet Child O Mine Audio');
% p8 = audioplayer(y,Fs);
% playblocking(p8);

% Plot fft to identify the guitar
subplot(2,1,2);
plot(ks,abs(fftshift(Yt))/max(abs(Yt)), 'blue');
set(gca, 'Xlim', [0 2e3]);
xlabel('frequency (Hertz)')
ylabel('Amplitude');
title('Fast Fourier Transform');

sgtitle('GNR Audio and FFT')
saveas(gcf,'GNR_Audio_FFT.jpg')
%close(gcf);

%% Testing different window widths for GNR
a = [1,10,100]; %compare across different window sizes.
tau = 0:0.1:L;

figure(2)

for jj = 1:length(a)
    Ygt_spec = []; % Clear at each loop iteration
    for j = 1:length(tau)
        g = exp(-a(jj)*(t - tau(j)).^2); % Window function
        Yg = g.*Y;
        Ygt = fft(Yg);
        Ygt_spec(:,j) = fftshift(abs(Ygt));
    end
    subplot(2,2,jj)
    pcolor(tau,ks,Ygt_spec)
    shading interp
```

```
    set(gca,'Ylim',[0,2e3],'Fontsize',16)
    colormap(hot)
    %colorbar
    xlabel('time (t)'), ylabel('frequency (k)')
    title(['a = ',num2str(a(jj))],'Fontsize',16)
end

%We observed the tradeoff & decide to use window size a=100
%% Testing different tau (oversampling and undersampling)

tau = 0:0.3:L;
%tau1 = 0:1:L;
%tau001 = 0:0.01:L;
Ygt_spec = [];

for j = 1:length(tau)
    g = exp(-100*(t - tau(j)).^2); % Window function
    Yg = g.*Y;
    Ygt = fft(Yg);
    Ygt_spec(:,j) = fftshift(abs(Ygt));
end

figure(3)
pcolor(tau,ks,Ygt_spec)
%pcolor(tau1,ks,Ygt_spec)
%pcolor(tau001,ks,Ygt_spec)

shading interp
title('Normalsampling tau=0.3')
set(gca,'Ylim',[0,2e3],'Fontsize',16)
xlabel('time (t)'), ylabel('frequency (Hz)')
colormap(hot)
%We observed the tradeoff & decide to use tau=0.3
```

```
%% Question 1: Reproduce the music score for GNR

%for the guitar in GNR
t_guitar = t2(1:n);
L_guitar = L;
k_guitar = (1/L_guitar)*[0:(n/2 - 1) -n/2:-1];
ks_guitar = fftshift(k_guitar);

tau_guitar = 0:0.3:L_guitar;
spec_guitar = [];
notes_guitar = [];
for j = 1:length(tau_guitar)
    g_guitar = exp(-100*(t_guitar - tau_guitar(j)).^2);
    Yg_guitar = g_guitar.*Y;
    Ygt_guitar = fft(Yg_guitar);
    [M,I] = max(Ygt_guitar); %find frequency centre
    notes_guitar(1,j) = abs(k_guitar(I)); %we don't want to rescale it
    spec_guitar(:,j) = fftshift(abs(Ygt_guitar));
end

figure(4)
plot(tau_guitar, notes_guitar, 'o','MarkerFaceColor','b');
title('Music score for the guitar in the GNR clip');
yticks([274,311,370,415,555,700,741]);
yticklabels({'C#','D#','F#','G#','C#','F','F#'});
ylim([200 900])
xlabel('time (t)'), ylabel('frequency (Hz)')

%% Comfortable Numb Isolating Bass
[y_bass, Fs_bass] = audioread('Floyd.m4a');
Y_bass = y_bass';
Yt_bass = fft(Y_bass);
```

```
n_bass = length(Y_bass); % Fourier modes
L_bass = n_bass/Fs_bass; % record time in seconds (time domain L)
t2_bass = linspace(0,L_bass,n_bass+1);
t_bass = t2_bass(1:n_bass);
k_bass = (1/L_bass)*[0:(n_bass/2 - 1) -n_bass/2:-1]; % Notice the 1/L instead of 2*pi/L
ks_bass = fftshift(k_bass);



figure(5)
% Plot audio
subplot(2,1,1);
plot(t_bass,Y_bass);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Comfortably Numb Audio');
% p8 = audioplayer(y,Fs);
% playblocking(p8);

% Plot fft to identify the bass
subplot(2,1,2);
dummy=abs(fftshift(Yt_bass))/max(abs(Yt_bass));

g = exp(-0.0008*(ks_bass - 155).^2);

plot(ks_bass,dummy(1:length(dummy)-1), 'blue');
hold on
plot(ks_bass,g,'m','Linewidth',2);

set(gca, 'Xlim', [0 2e3]);
xlabel('frequency (Hertz)')
ylabel('Amplitude');
title('Fast Fourier Transform');
sgtitle('Floyd Audio and FFT')
```

```
saveas(gcf,'Floyd_Audio_FFT.jpg')

%Again, we lost all the time information so we want to use Gabor transform
%instead

%% Divide Floyd audio into 3 pieces
y1 = y_bass(1:length(y_bass)/3);
y2 = y_bass(length(y_bass)/3:2*length(y_bass)/3);
y3 = y_bass(2*length(y_bass)/3:length(y_bass));

%% Use a filter in frequency space to try to isolate the bass in Comfortably Numb (60-250hz);

% Testing different window widths
a_bass = [1,10,100]; %compare across different window sizes.
tau_bass = 0:1:L_bass;

figure(6)
%
% for jj = 1:length(a_bass)
%     spec_bass = []; % Clear at each loop iteration
%     for j = 1:length(tau_bass)
%         g = exp(-a_bass(jj)*(t_bass - tau_bass(j)).^2); % Window function
%         Yg_bass = g.*Y_bass;
%         Ygt_bass = fft(Yg_bass);
%         spec_bass(:,j) = fftshift(abs(Ygt_bass));
%     end
%     subplot(2,2,jj)
%     pcolor(tau_bass,ks_bass(60:250),spec_bass(60:250,:))
%     shading interp
%     set(gca,'Ylim',[60,250],'Fontsize',16)
%     colormap(hot)
%     %colorbar
%     xlabel('time (t)'), ylabel('frequency (k)')
```

```
%     title(['a = ',num2str(a_bass(jj))],'Fontsize',16)
% end
spec_bass=[];

for k = 1:length(tau_bass)
    g = exp(-10*(t_bass - tau_bass(k)).^2); % Window function
    Yg_bass = g.*y1;
    Ygt_bass = fft(Yg_bass);
    spec_bass(:,k) = fftshift(abs(Ygt_bass));
end

spec_bass=spec_bass(60:250,:);

pcolor(tau_bass,ks_bass(60:250),spec_bass)
shading interp
title('Normalsampling tau=1, a=100')
set(gca,'Ylim',[60,250],'Fontsize',16)
xlabel('time (t)'), ylabel('frequency (Hz)')
colormap(hot)

%We observed the tradeoff & decide to use window size a=100
%% Floyd Bass Note (60-250hz)
tau_bass = 0:0.3:L_bass;
spec_bass = [];
notes_bass = [];
for j = 1:length(tau_bass)
    g_bass = exp(-100*(t_bass - tau_bass(j)).^2);
    Yg_bass = g_bass.*Y_bass;
    Ygt_bass = fft(Yg_bass);
    [M,I] = max(Ygt_bass); %find frequency centre
    notes_bass(1,j) = abs(k_bass(I)); %we don't want to rescale it
    spec_bass(:,j) = fftshift(abs(Ygt_bass));
end
```

```
figure(4)
plot(tau_bass, notes_bass, 'o','MarkerFaceColor','b');
title('Music score for the bass (60~250hz) in the Floyd clip');
yticks([82,91,97,110,123,165,185,196,246]);
yticklabels({'E(82)','F#(91)','G(97)','A(110)','B(123)','E(165)','F#(185)','G(196)','B(246)'});
ylim([60 250])
xlabel('time (t)'), ylabel('frequency (Hz)')

%% Floyd Guitar (60-250hz)
spec_guitar = [];
notes_guitar = [];
for j = 1:length(tau_bass)
    g_bass = exp(-100*(t_bass - tau_bass(j)).^2);
    Yg_bass = g_bass.*Y_bass;
    Ygt_bass = fft(Yg_bass);
    [M,I] = max(Ygt_bass); %find frequency centre
    notes_guitar(1,j) = abs(k_bass(I)); %we don't want to rescale it
    spec_bass(:,j) = fftshift(abs(Ygt_bass));
end

figure(4)
plot(tau_bass, notes_guitar, 'o','MarkerFaceColor','b');
title('Music score for the guitar (300~800Hz) in the Floyd clip');
yticks([331,370,392,491,590,673,741]);
yticklabels({'E(331)','F#(370)','G(392)','B(491)','D(590)','E(673)','F#(741)'});
ylim([300 800])
xlabel('time (t)'), ylabel('frequency (Hz)')
```