Title: Rock & Roll and the Gabor Transform

Author: Wendy Jiang

Abstract: This is the assignment to analyze a portion of two rock & roll songs of all time. We import the .m4a file and convert them into vectors to represent the music. Thus, we could easily edit the music by modifying the vector. Then we use Gabor filtering to reproduce the music score by seeing the spectrogram. To get the better result, we would filter the overtones.

Section I. Introduction and Overview

This report aims to explain the process of analyzing the portion of songs and find the frequency spectrogram in time domain to locate the music score based on the Gabor Transform. To make the result more precisely, filter the overtone and separate the space of bass and guitar depend on the different frequency space. Here we use two songs clips: *Sweet Child O' Mine* by Guns N' Roses and *Comfortably Numb* by Pink Floyd.

Section II. Theoretical Background

1. The Fourier Transform has the drawback that the shift invariance of the absolute value of the Fourier Transform loses information about what is happening in the time domain. Particularly, we lost when certain frequencies occur or how the frequencies change over time. To get both time and frequency information from a signal, we split up time domain into subdomains, and then take the Fourier Transform on each subdomain. This method could capture the fact that there are higher frequencies in the beginning and lower frequencies in the middle, but it exists some problems in practice for a few reasons: windows are fixed so we do not have information about portions of signal that span multiple windows; there are sharp transitions between regions, which potentially can cause problems. To improve the method, we can use time-filter to pick out a window. This is the basis of the Gabor Transform, also called the short-time Fourier Transform.

2. Shifting by τ and multiply by a function $f(t)$ represents the filter function $f(t)g(t-\tau)$.

   The Gabor Transform:

   $$\widetilde{f_g}(\tau,k) = \int_{-\infty}^{\infty} f(t)g(\tau-t)e^{ikt}dt. \quad ...(1)$$

   The function $\widetilde{f_g}(\tau,k)$ gives the information about the frequency components near time τ. The result is dependent on the choice of filter $g(t)$, hence the subscript g on $\tilde{f}$.

3. The inverse of the Gabor Transform

   $$f(t) = \frac{1}{2\pi||g||_2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\widetilde{f_g}(\tau,k)g(t-\tau)e^{ikt}dkd\tau. \quad ...(2)$$

4. If the width of the Gabor window is huge, then we can just cover the Fourier Transform over the whole signal, which has all the frequency information, but no information about

time. If the window is extremely small, when we are almost look at the individual times along the signal, and so there would be no frequency information.

5. Discrete Gabor Transform: The most important aspect of this is that we cannot shift the window by any arbitrary τ. It can only be taken in some discrete set of values. $k = m\omega_0$, $\tau = nt_0$. The discrete Gabor Transform function:

$$\widetilde{f_g}(m,n) = \int_{-\infty}^{\infty} f(t)g(t-nt_0)e^{2\pi i m\omega_0 t}dt. \quad ...(3)$$

6. The Gabor Transform can visualize things in the time-frequency domain. We are just multiple the signal by a window function or a filter function and then apply the FFT. The window would be moved and repeat the process. The Gaussian window function: $g(t-\tau) = e^{-a(t-\tau)\wedge 2}$ ...(4)
Two parameters: τ and $a > 0$, representing the center of the window and the width of the window. The small a means a wide window and vice versa, the large a means a thin window.

7. The MATLAB has its own built-in spectrogram function that makes uses of the Gabor Transform.
We only need to plot positive frequencies, but this is not the arbitrary MATLAB decision. Since the signal are real values, only the positive frequencies are needed since the coefficient on the terms with negative frequencies are jest the complex conjugate of the corresponding positive frequency.
MATLAB automatically scales the frequencies so that the max is 1. We need to get rid of it.

8. MATLAB scales out the 2π in the period of oscillation to makes the period integers. That is,

$$e^{ikt}, \qquad k \in \mathrm{Z}. \quad ...(5)$$

has frequency $\frac{|k|}{2\pi}$, while

$$e^{i2\pi ft}, \qquad f \in Z. \quad ...(6)$$

has frequency $|f|$.
Equation (5) and (6) both represent frequencies, but they have different units. With the notation above, f is measured in Hertz, while k is sometimes called the angular frequency. Therefore, if we go back and rescale out k in the spectrogram by $1/L$ instead by $2\pi/L$ we will get the same thing.

9. Overtone: a musical tone which is a part of the harmonic series above a fundamental note and may be heard with it.

Section III. Algorithm Implementation and Development

In image processing, a Gabor filter, which essentially means that it analyzes whether there is any specific frequency content in the image in specific directions in a localized region around the

point or region of analysis. Here, the Gabor Filter could help us analysis song's clips. Our goal for this assignment is to get the spectrogram of the song's clip's signal. To achieve our goal, we use the Gabor Transform to get the relationship between time domain and frequency. To build the window, we need a parameter called a to represent its width and here, for the song *Sweet Child O' Mine*, we set $a = 1000$. For a bigger range of time domain, since we have much more music scores, we need a more localized window, here we set $a = 6000$ in the transform of the *Comfortably Numb*. And τ represent the center of the window. We use τ = 0.1 of the *Sweet Child O' Mine* and τ = 1 of the *Comfortably Numb*. Hence the Gabor transform is similar to the Fourier Transform except using the time window. As the same process with Fourier Transform, we set the frequency domain k, by using spatial domain L and the function $1/L$ in this assignment. To keep the window moving we set a for loop. Inside the loop, first the window function is defined: $e^{-a(t-\tau)^2}$. To apply the Gabor Transform, we multiply window function with the signal and then use Fourier transform, keep the positive part and reorder it.

Function $y = fft(x)$ computes the discrete Fourier Transform (DFT) of x using FFT algorithm. "The FFT functions (fft, fft2, fftn, ifft, ifft2, ifftn) are based on a library called FFTW [1] [2]." (form https://www.mathworks.com/help/matlab/ref/fft.html#buuutyt-11)
Function $x = ifft(y)$ computes the inverse discrete Fourier Transform using FFT algorithm. "The ifft function tests whether the vectors in Y are conjugate symmetric. A vector v is conjugate symmetric when it equals conj(v([1,end:-1:2])). If the vectors in Y are conjugate symmetric, then the inverse transform computation is faster, and the output is real." (from https://www.mathworks.com/help/matlab/ref/ifft.html)

Section IV. Computational Result

Figure1 and Figure3 are the spectrogram of the two song's clip without removing overtones. Hence, we could see many blurry wavelets on the plot.

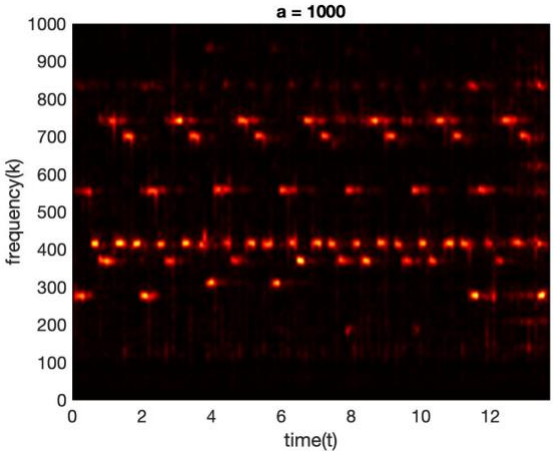

Figure1. The spectrogram of the *Sweet Child O' Mine*

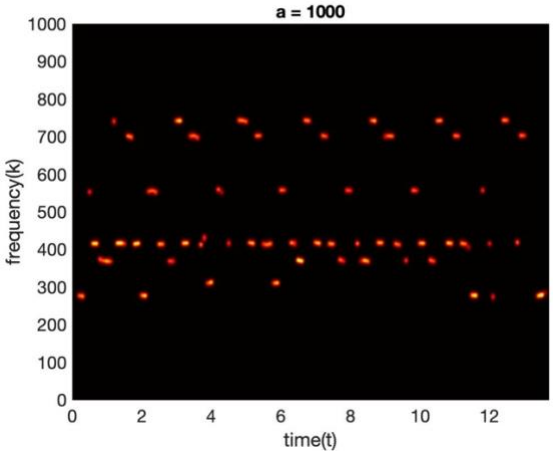

Figure2. The spectrogram of the *Sweet Child O' Mine after removing overtones*

Figure2 is the spectrogram of time and frequencies after filtering the overtones of GNR, which using the Gaussian filter to clear the white noise. Of the Gaussian filter function we set the index of the function equals 0.01. Figure1 represents all guitar scores. The lighter spot means the clearer tone, or we can say it has the higher amplitude.

According to the frequency of the music score, we could approximate the music notes of this clip of *Sweet Child O' Mine* here: C#(277) C#(554) G#(415) F#(370) F#(740) G#(415) F(698) G#(415) (repeat one more time); D#(311) C#(554) G#(415) F#(370) F#(740) G#(415) F(698) G#(415) (repeat one more time); F#(370) C#(554) G#(415) F#(370) F#(740) G#(415) F(698) G#(415) (repeat one more time); C#(277)

(Note: We use # to represent halftone sharper. For example, C# is halftone shaper than C. And inside the parentheses is the frequency of this score.)

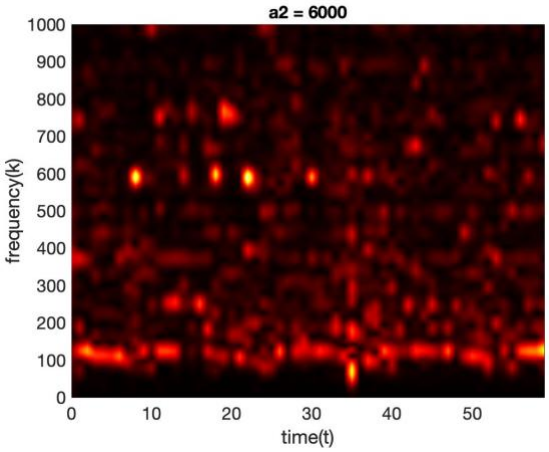

Figure3. The spectrogram of both bass and guitar *in Comfortably Numb* before clear overtones

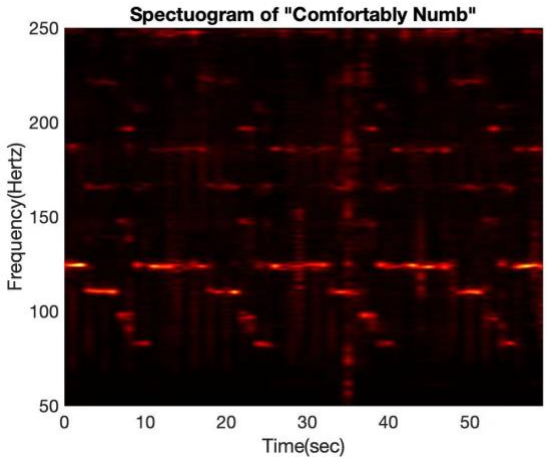

Figure4. The spectrogram of both bass *in Comfortably Numb* before clear overtones

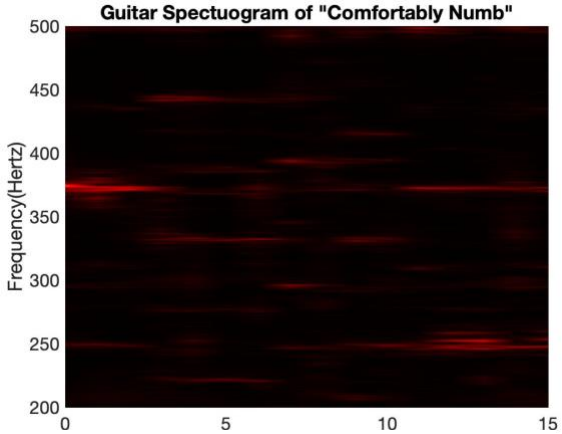

Figure5. The spectrogram of guitar of the Comfortably Numb

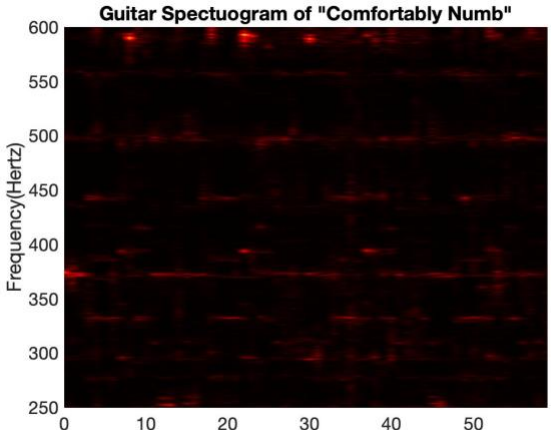

Figure6. The spectrogram of the guitar in the *Comfortably Numb when we set smaller a*

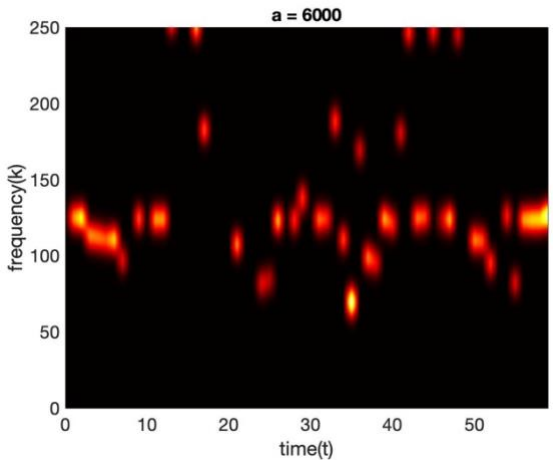

Figure7. The spectrogram of the *bass in the Comfortably Numb after removing overtones*

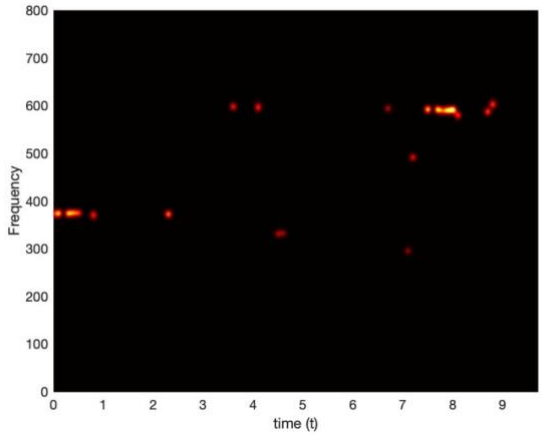

Figure8. The spectrogram of the guitar in the Comfortably Numb *after removing overtones*

Figure3 is the spectrogram of the whole 60 sec clip of the bass part Floyd's song Comfortably Numb. Here we also remove the overtone by using Gaussian filter in order to get clear plot of guitar and bass area below.

Based on the knowledge of music, bass have lower frequency than guitar (frequency between 50 – 250). Hence, by Figure4 and Figure7 We can figure out the bass part of Comfortably Numb is almost repeat the period of B(123) A(110) G(97) F(87) E(82) over time: B B A A A G F E B …. There is repeat two times of this period in total 60 sec.

By watching Figure5, Figure6 and Figure8, as we know, guitar's tone has the frequency over 250. We could limit the time domain and use a smaller a = 1000 here to build a wider window to see the details, so need use smaller τ = 0.1. We could simplify find the group of music notes:

F#(370) F#(370) F#(370) F#(370) D(587) D(587) D(294) D(587) D(294) B(494) D(587) D(587) D(587) C#(554) D(587) D#(622) ….

Section V. Summary and Conclusions

To find the music notes of song's clips, the Gabor filter is the good tool to transfer signals and get plot of frequency into time domain. The Gaussian filter enable to reduce overtone and help to get a clearer and more precise spectrogram. And according the basic rule that each music note has various frequency, we could reproduce the music score of the clips.

Appendix A. MATLAB functions used and brief implementation explanation

- fftshift(): rearrange a Fourier Transform x and shifting zero frequency component to the center of the array.

- ifftshift(): rearrange a zero-frequency-shifted Fourier Transform back to the original transform output.
- linespace($x1, x2, n$): this functions is used to generate points between two values.
- fft: returns the array using Fourier transform algorithm.
- Ifft(): returns the discrete inverse Fourier transform of an array using FFT algorithm.
- [y,Fs] = audioread(filename) reads data from the file named filename, and returns sampled data, y, and a sample rate for that data, Fs.
- pcolor(X,Y,C) specifies the x- and y-coordinates for the vertices. The size of C must match the size of the x-y coordinate grid. For example, if X and Y define an m-by-n grid, then C must be an m-by-n matrix.

Appendix B. MATLAB Code

```
%%
clear all; close all; clc

%% get the signal plot of GNR
figure(1)
figure(1)
[y, Fs] = audioread('GNR.m4a');
tr_gnr = length(y)/Fs; % record time in seconds
plot((1:length(y))/Fs,y);
xlabel('Time [sec]'); ylabel('Amplitude');
title("'Sweet Child O' Mine");
p8 = audioplayer(y,Fs);playblocking(p8);

S = y';
k = (1/L)*[0:n/2-1 -n/2:-1]; %set the frequency domain k
ks = fftshift(k); % fourier transform of k
t2 = linspace(0,L,n+1); %discretization
t = t2(1:n);  % only the first n points (periodicity)

a = 1000; % set window scale
tau = 0:0.1:trgnr; % set window center
Sgt_spec = zeros(n, length(tau));

for j = 1:length(tau)
    g = exp(-a*(t - tau(j)).^2); % Window function
    Sg = g.*S; % apply window funtion
    Sgt = fft(Sg); % fourier transform
    [m, ind] = max(abs(Sgt)); % get the index
    Sgtf = Sgt .* exp(-0.01 * (k - k(ind)).^2); % filter overtone
    Sgt_spec(:, j) = fftshift(abs(Sgtf));
end
```

```
figure(2)
pcolor(tau, ks, Sgt_spec)
shading interp
colormap(hot)
set(gca,'ylim',[0 1000],'Fontsize',16)
xlabel('time(t)'),ylabel('frequency(k)')
title('a = 1000','Fontsize',16)



%% get the signal plot of floyd

figure(2)
[y2, Fs2] = audioread('Floyd.m4a');
tr_gnr2 = length(y2)/Fs2; % record time in seconds
plot((1:length(y2))/Fs2,y2);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Comfortably Numb');
p = audioplayer(y2,Fs2); playblocking(p);

S = y';
k = (1/L)*[0:n/2-1 -n/2:-1]; %set the frequency domain k
ks = fftshift(k); % fourier transform of k
t2 = linspace(0,L,n+1); %discretization
t = t2(1:n);  % only the first n points (periodicity)


tau = 0:1:trgnr; % set window center
a = 6000; % set window scale
Sgt_spec = zeros(n - 1, length(tau));

for j = 1:length(tau)
    g = exp(-a * (t - tau(j)).^2); % Window function
    Sg = g .* S;
    Sgt = fft(Sg);

    Sgt = Sgt(1:n-1);
    [m, ind] = max(abs(Sgt));
    filter = exp(-0.01 * (k - k(ind)).^2); % define filter function
    Sgtf = Sgt .* filter; % filter overtone
    Sgtf(k > 250) = 0; % filter out any frequency higher than 250
    Sgt_spec(:, j) = fftshift(abs(Sgtf));
end
figure(3)
pcolor(tau, ks, Sgt_spec);
shading interp
colormap(hot);
set(gca,'ylim',[0,250],'Fontsize',16)
xlabel('time(t)'),ylabel('frequency(k)')
title('a = 6000','Fontsize',16)
%%
% reset window center and scale
tau = 0:0.1:trgnr;
a = 1000;
S = y';
```

```
Sgt_spec = zeros(n-1, length(tau));

for j = 1:length(tau)
    g = exp(-a*(t - tau(j)).^2); % Window function
    Sg = g.*S;
    Sgt = fft(Sg); % fourier transform
    Sgt = Sgt(1:n-1); % reset dimension
    [maximum, index] = max(abs(Sgt));
    filter = exp(-0.01*(k-k(index)).^2); % define Gaussian filter function
    Sgtf = Sgt .* filter; % filter overtone
    bass_note = Sgtf;
    bass_note(k > 250) = 0; % filter out any frequency higher than 250
    guitar_note = Sgtf - bass_note;
    Sgt_spec(:, j) = fftshift(abs(guitar_note));
end

figure(4)
colormap(hot)
pcolor(tau,ks,Sgt_spec)
shading interp
set(gca,'ylim',[0 800],'Fontsize',12)
colormap(hot)
xlabel('time (t)'), ylabel('Frequency')
```