AMATH 482 Homework 2:
Rock & Roll and the Gabor Transform

Malikah Nathani
February 10th, 2021

Abstract

This paper examines the use of the Gabor transform in time-frequency music analysis by evaluating the signals given in two Rock & Roll songs. Specifically, by extracting the notes played by different instruments. Furthermore, by combining the technique of Gaussian filtering, we can omit overtones present to ultimately identify the unique instruments played.

1 Introduction and Overview

Time-frequency analysis is a vast area of study with numerous applications, including music signals which will be the focus of this paper. As music signals are time-varying, the Gabor transform is frequently implemented to obtain time and frequency information about the signal. For analyzing music, it is important to note that each musical sound has two parts: fundamental frequency and overtones. The fundamental frequency is the true pitch of the note played while overtones are integer multiples of the fundamental frequency that add harmonics to produce a more pleasing sound. The scale used for musical note frequencies is presented in Hertz.

This paper focuses on a specific problem: analyzing music signals to derive the notes played. Given the audio clips of the guitar solos for two Rock & Roll songs, *Sweet Child O' Mine* by Guns N' Roses and *Comfortably Numb* by Pink Floyd, we are tasked to find the notes for the guitar and bass respectively. Afterward, we are asked to filter the Pink Floyd song in frequency space to strip *Comfortably Numb* of the overtones present and isolate the bass as well as attempt to identify the guitar solo.

In Section 2, I will go about explaining the Gabor transform in more detail while section 3 will go over the implementation of this method and the algorithm used to solve the problem. Sections 4 and 5 will go over the computed results and my conclusions respectively. Finally, I will also present the MATLAB functions used and provide my code.

2 Theoretical Background

2.1 Gabor Transform

For this application of time-frequency analysis, it is vital to obtain time and frequency information about a certain signal, thus we utilize the Gabor transform. The idea behind this method is to break the Fourier transformed signal into subdomains, we use a time-filter to pick a

window and sweep it across to domain to consider all possible locations of the filter thus obtaining frequency and time information. There are two main components of the filter, τ describing where the filter is centered and the value a describing the width of the subdomain we are considering at that time point.

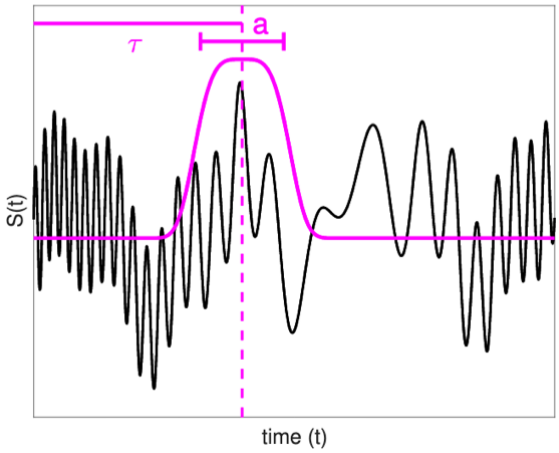


Figure 1 Visualization of Gabor transform components

The Gabor transform formula is given by:

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

Where $f(t)g(t-\tau)$ represents a function $f(t)$ multiplied by a filter function $g(t)$ shifted by τ, $g(t-\tau)$. For a fixed τ, $\tilde{f}_g(\tau, k)$, gives information of the frequencies near time τ. It is important to note that results are dependent on the choice of $g(t)$, however, common assumptions to note about $g(t)$ are:

1. The function is symmetric and real.
2. The L2-norm of $g(t)$ is set to unity.

In this problem, we will use a Gaussian filter. This filter allows us to keep the signal information in our a window and get rid of everything else for that window. By shifting our filter across the domain, we will eventually get little windows for the whole signal ultimately giving us frequency and time information. The inverse of the Gabor transform is given by:

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

An important aspect to consider, when applying $g(t)$, is how big the width of the subdomain, a, should be. If the window is too large, then the whole Fourier transformed signal is recovered therefore giving no information on time. However, if the a value is too small, we become too localized in time and lose information about the frequencies. This is the Heisenberg uncertainty principle at play, which states, in context of this problem, the more known about time the less is known about the frequency of the signal and vice versa. Therefore, it is important to strike a balance of the width to get some information about both time and frequency.

2.1.1 Discrete Gabor Transform (DGT)

The discrete Gabor transform is a method to determine the Gabor transform of a signal given discrete data points. Most importantly, when using the DGT an arbitrary τ cannot be used

to shift the window, it can only be taken in a given discrete set of values. Another aspect to consider is a discrete set of frequencies given by:

$$k = mw_0$$
$$\tau = nt_0$$

Where m and n are integers while w_0 and t_0 are positive constants. Therefore, the DGT is given as:

$$\tilde{f}_g(m,n) = \int_{-\infty}^{\infty} f(t)g(t-nt_0)e^{2\pi imw_0 t}dt$$

To effectively reproduce a signal and have visible frequency resolution, it is important to have a window large enough or a small enough time step t_0 to have some overlap, as well as have a small enough w_0.

3 Algorithm Implementation and Development

This section details the implementations of the methods above and any other steps taken to solve this problem in MATLAB. For the actual code used, please see Appendix B.

First, we need to load in our song clips to obtain the sampled data and sampled data rate for the song. We also need to define the spatial domain L (length of domain) and the number of Fourier modes n. One aspect to consider in implementation, is that the FFT command in MATLAB assumes 2π periodic signals ($-\pi$, π). Since our frequencies are on the scale of (-L, L), before using the FFT functions in MATLAB it is important to rescale all the frequencies by $\frac{2\pi}{L}$. However, since the scale used for musical note frequencies are given in Hertz (Hz), we instead scale our frequencies by $\frac{1}{L}$ and then shift such that the 0-component frequency is in the center of our array.

Afterward, we determine our a and τ values to be able to use the Gabor transform and loop over the signal. For this problem, we keep a constant a value of 100 across both songs. However, our step size differs between *Sweet Child O' Mine* and *Comfortably Numb* since the clip lengths differ and a smaller step size takes more time and computer memory. For *Sweet Child O' Mine* we use a step size of 0.1, while in *Comfortably Numb* our value is 0.5. We multiply our filter to the sampled data to create a window, then use the Fourier transform to shift it into the frequency domain. Then, we add the values obtained to a matrix to act as our third dimension to understand which frequency is the fundamental one. We repeat this process for all the created windows in the domain.

To plot our results, we use a contour plot where time is on the x-axis and our shifted frequencies on the y-axis. The matrix is used as a third dimension to understand where the high and low values are, it is important to note that we do not show the values relative to each other to get a better understanding of the data present.

When plotting our Guns N' Roses song, although there are overtones present, we get a clear view on the fundamental frequencies thus can easily extract the musical notes for the guitar. However, for the Pink Floyd song, it is hard to make out the bass notes specifically. With our basic music theory understanding, we know that the bass has a very low frequency, therefore, when plotted, we can add limits to the y-axis from [0, 300] inclusive to get a better view of the bass frequencies. However, another way to clearly see the notes played is by removing the overtones present. This process can also be done by a Gaussian filter.

To remove the overtones, we use the same basic process as above, but we add a few steps. After shifting the windowed signal into the frequency space, we use built in MATLAB functions (described in detail in Appendix A) to obtain the index of the maximum absolute-valued frequency.

This index will be that of our central frequency. Then, we multiply our signal by the Gaussian filter to dissolve any noise. We will be left with the fundamental frequency which will be the true note of the bass, as that would have been the strongest (max) frequency. For this filter, we used a τ value of 0.2. The higher the τ value, the wider the filter thus the more information along the sides of the center frequency is let it. You want a width where not too much information is coming in, as then it may contain all the noise still present in the signal. However, a filter width too small is also undesirable, as that may not contain enough information to give you a proper understanding of the data and what it represents.

Finally, to determine the guitar notes played, we used a portion of the clip between timestamp [30,40] seconds as this is where the bass notes seemed to be very low; we used a timestep value of 0.1 as this was a smaller clip. We used the same process as above to get the frequencies of the song over time without the overtones. With our, knowledge of instrument frequencies, we can determine that the fundamental frequencies of the bass cannot be above a certain Hz, therefore by limiting the y-axis from [300,1000], we can presume that the notes shown are those of the guitar solo. Although this method is more speculative, with more time and resources we may be able to develop a path to further explore this concept; ultimately, we can piece together the whole guitar solo.

4 Computational Results

Below show the results from using the method above. Please note, when plotting our third dimension we used $\log(|s| + 1)$ where s represents our spectrogram.

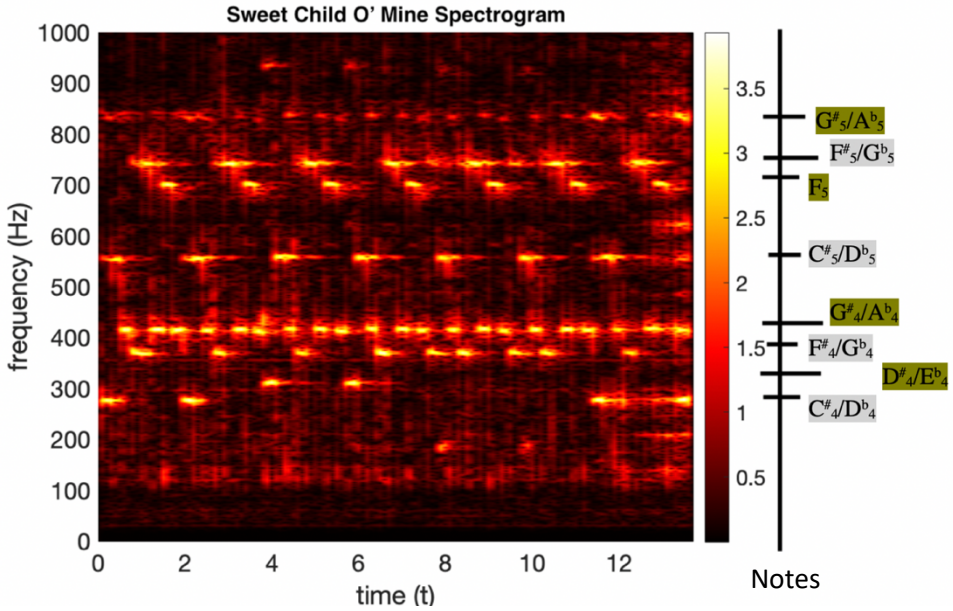
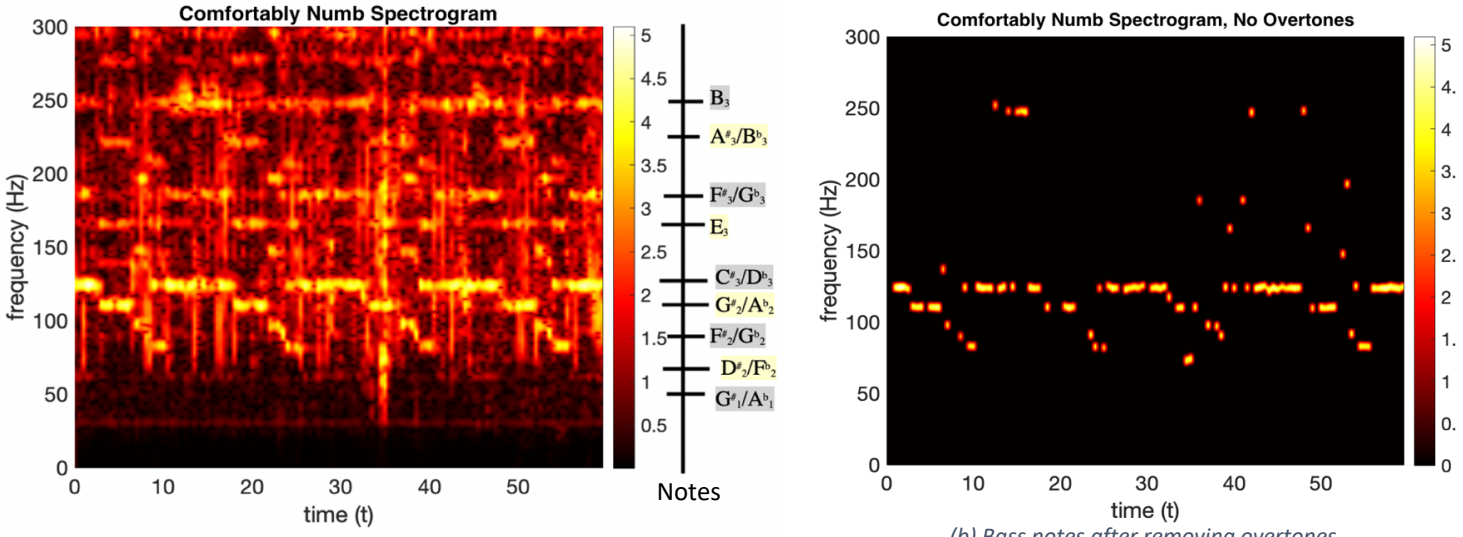


Figure 2 Spectrogram to show the guitar notes for Sweet Child O' Mine, notes axis is labelled to the right



(a) Bass notes without filtering overtones

(b) Bass notes after removing overtones

Figure 3 Spectrogram to show the bass notes for Comfortably Numb before and after accounting for overtones, notes axis is labelled in the center

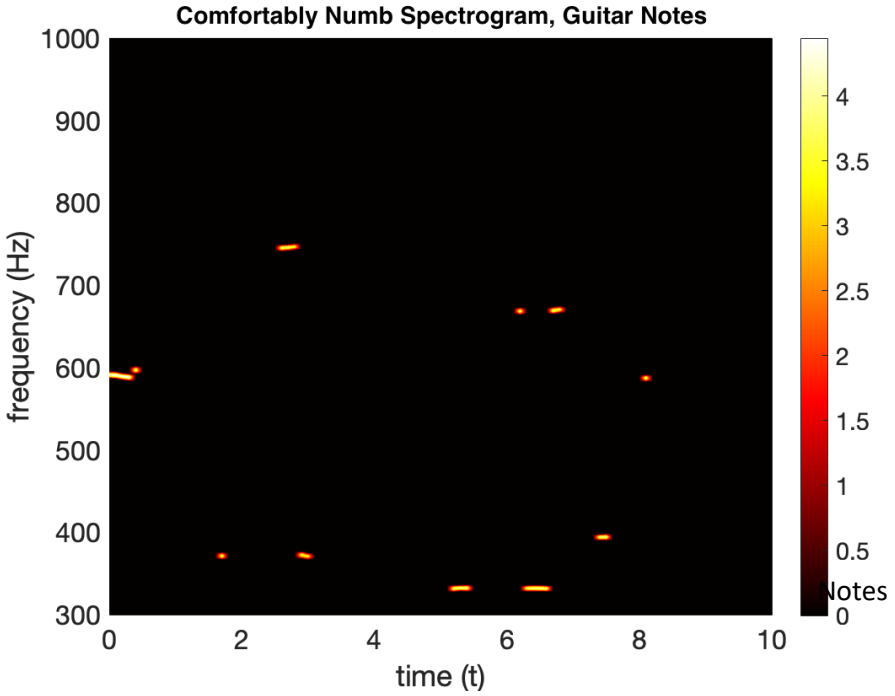


Figure 4 Spectrogram to show the possible guitar notes for Comfortably Numb

5 Summary and Conclusions

Through the use of the Gabor transform, we successfully obtained the musical notes played by the guitar for *Sweet Child O' Mine* and the bass for *Comfortably Numb* while easily removing the overtones with a Gaussian filter. We also attempted to identify the guitar solo for a portion of the Pink Floyd song. With further study of time-frequency analysis, we may be able to identify a path to clearly derive the guitar notes for the whole audio clip. Therefore, showing

how understanding mathematical methods and their applications is an ongoing process and has limitless opportunity.

Appendix A. MATLAB functions used

- [y, Fs] = audioread(filename): reads data from the file named filename, and returns sampled data, y, and a sampled rate for that data, Fs
- plot(X, Y): creates a 2-D line plot of the data in Y versus the corresponding values in X
- y = linspace(x1, x2, n): generates a row vector of n evenly spaced points between x1 and x2
- Y = fftshift(X): rearranges a Fourier transform X by shifting the zero-frequency component to the center of the array
- Y = fft(X): computes the discrete Fourier transform of X using a fast Fourier transform algorithm
- pcolor(X, Y, C): creates a pseudocolor plot using the values in the matrix C, while X and Y specify the coordinates for their respective vertices
- [M, I] = max(X): returns the max value as well as the corresponding index in the operating dimension

Appendix B. MATLAB codes

```
%% Set up
clear; close all; clc;

%% Import Floyd Song
figure(1)
[yf, Fs] = audioread('Floyd.m4a');
tr_pf = length (yf) / Fs; % record time in seconds
plot((1:length(yf)) / Fs, yf);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Comfortably Numb');
% p8 = audioplayer(yf, Fs); playblocking(p8);

%% Import GNR Song
figure(2)
[yG, Gs] = audioread('GNR.m4a');
tr_gna = length (yG) / Gs; % record time in seconds
plot((1:length(yG)) / Gs, yG);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Sweet Child O'' Mine');
p9 = audioplayer(yG, Gs); playblocking(p9);

%% Reproduce Music Score for GNR Song
L = tr_gna;
n = length(yG);
t2 = linspace(0,L,n+1);
t = t2(1:n);

k = (1/L)*[0:n/2-1 -n/2:-1];
ks = fftshift(k);
```

```
a = 100;
tau = 0:0.1:L;

for j = 1:length(tau)
    g = exp(-a*(t-tau(j)).^2); % Window function
    Sg = g'.* yG;
    Sgt = fft(Sg);
    Sgt_spec(:,j) = fftshift(abs(Sgt));
end

figure(3)
pcolor(tau, ks, log(abs(Sgt_spec+1)))
shading interp
set(gca, 'ylim', [0, 1000],'Fontsize', 16)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')

%% Reproduce Music Score for PF Song
L = tr_pf;
n = length(yf);
t2 = linspace(0,L,n+1);
t = t2(1:n);

k = (1/L)*[0:n/2-1 -n/2:-1];
ks = fftshift(k);

a = 100;
tau = 0:1:L;

for j = 1:length(tau)
    g = exp(-a*(t-tau(j)).^2); % Window function
    Sg = g'.* yf;
    Sgt = fft(Sg);
    Sgt_spec(:,j) = fftshift(abs(Sgt));
end

Sgt_spec = Sgt_spec(1:length(ks), :);

figure(4)
pcolor(tau, ks, log(abs(Sgt_spec + 1)))
shading interp
set(gca, 'ylim', [0, 300],'Fontsize', 16)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')

%% Part 2, taking out the overtones of the bass

L = tr_pf;
n = length(yf);
t2 = linspace(0,L,n+1);
t = t2(1:n);
k = (1/L)*[0:n/2-1 -n/2:-1];
ks = fftshift(k);
```

```
a = 100;
tau = 0:0.5:L;

ti = 0.2;

for j = 1:length(tau)
    g = exp(-a*(t-tau(j)).^2); % Window function
    Sg = g'.* yf;
    Sgt = fft(Sg);
    [M, I] = max(abs(Sgt));

    Sgt = Sgt(1:length(ks), :);
    Sgtft = Sgt'.* exp(-ti*(k-k(I)).^2);

    Sgt_spec(:,j) = fftshift(abs(Sgtft));
end

figure(5)
pcolor(tau, ks, log(abs(Sgt_spec+1)))
shading interp
set(gca, 'ylim', [0, 300],'Fontsize', 16)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')

%% Part 3
% Importing shortened track
[yf, Fs] = audioread('Floyd2.m4a');
tr_pf = length (yf) / Fs; % record time in seconds
plot((1:length(yf)) / Fs, yf);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Comfortably Numb');
p8 = audioplayer(yf, Fs); playblocking(p8);

% Finding guitar notes
L = tr_pf;
n = length(yf);
t2 = linspace(0,L,n+1);
t = t2(1:n);
k = (1/L)*[0:n/2-1 -n/2:-1];
ks = fftshift(k);

a = 100;
tau = 0:0.1:L;

ti = 0.2;

for j = 1:length(tau)
    g = exp(-a*(t-tau(j)).^2); % Window function
    Sg = g'.* yf;
    Sgt = fft(Sg);
    [M, I] = max(abs(Sgt));

    Sgt = Sgt(1:length(ks), :);
    Sgtft = Sgt'.* exp(-ti*(k-k(I)).^2);
```

```
    Sgt_spec(:,j) = fftshift(abs(Sgtft));
end

figure(4)
pcolor(tau, ks, log(abs(Sgt_spec + 1)))
shading interp
set(gca, 'ylim', [300, 1000],'Fontsize', 16)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')
```