AMATH 482 Homework 2

Jerry Wen

January 24, 2021

Abstract

Here, we are examining two audio files. One contains a 13 second clip from Gun N' Roses' *Sweet Child O' Mine*. The second is a minute long clip from Pink Floyd's Comfortably Numb. We will want to find the music scores of each just by looking at the frequency data of each audio file.

1 Introduction and Overview

An audio file gets converted into a vector in MATLAB that we can filter using methods we learned like the Fourier Transform and the new Gabor Transform. The Fourier Transform is used to bring us into frequency space so we can examine the frequency level of the sound in Hertz to be able to determine what note it is. The Gabor transform is used to help us filter specific areas as it will give us a bit of information on both the frequencies that happened at a specific time, something that a normal Fourier Transform is unable to do as we leave the time domain.

1.1 The Guitar Score of Gun N' Roses

The first piece we examine is from Gun N' Roses as their opening guitar riff is short and loud, which makes it easier to detect with our current methods.

1.1.1 The Bass Score of Floyd

Our second objective is to create the Bass score from Floyd. A bass is much deeper than a guitar and thus is much harder to hear, which makes us have to use some creative strategies to only filter the bass out so it looks clean.

1.1.2 The Guitar Score of Floyd

Finally, we want to create the Guitar score from Floyd, however the bass score that we examined earlier has overtones which are frequencies that show up at frequencies 2x,3x,4x....(integer multiplications) above the fundamental frequency.

2 Theoretical Background

Our project revolves around manipulationg the frequency space. We will be using the Fourier Transform: If we are given a vector of N values from a function sampled at equally-spaced points $\{x_0, x_1, ..., x_N\}$, we can find the discrete Fourier Transform of it as:

$$\hat{x}_k = \frac{1}{N}\sum_{n=0}^{N-1} x_n e^{\frac{2\pi i k n}{N}} \tag{1}$$

In MATLAB, we use the Fast Fourier Transform which is the DFT but just faster. It's a divide and conquer algorithm that is already built into MATLAB which makes it much easier to use.

Another important method we use is the Gabor Transform or short-time Fourier transform: Consider a filter function g(t). Shifting by τ and multiplying a function f(t) represents the filtered function $f(t)g(t-\tau)$, with the filter centered at τ. The Gabor transform is then given by:

$$\tilde{f}_g(\tau, k) = \int_{-\infty}^{\infty} f(t)g(t-\tau)e^{-ikt}dt \tag{2}$$

This gives us information about the frequency components near time τ, which allows us to connect frequency with time, something that we do not normally get from the FFT.

Like the Fourier Transform having the Discrete Fourier Transform, we need a Discrete Gabor Transform as our audio file does not have an infinite domain. To do this, we have to consider a discrete set of frequencies:

$$k = m\omega_0 \tag{3}$$

$$\tau = nt_0 \tag{4}$$

Where ω_0 and t_0 are positive constants and m and n are integers. The discrete Gabor transform is:

$$\tilde{f}_g(\tau, k) = \int_{-\infty}^{\infty} f(t)g(t-nt_0)e^{2\pi i m\omega_0 t}dt \tag{5}$$

The Gabor transform has a constant that we can choose which selects the size of the window that we look at. A smaller constant gives us a wider window which allows us to have better accuracy in the frequency domain, but less accuracy in the time domain while a larger constant gives a skinnier window that allows for better time domain accuracy but worse frequency domain accuracy. For our implmentations, we experiment around with different constants to find one that gives the cleanest spectrogram.

Finally, we need to use the Gaussian so we can filter around our maximum value at each $t = \tau$ so our spectrograms can look cleaner, where it looks like:

$$F(k) = e^{-\tau(k-k_0)^2} \tag{6}$$

3 Algorithm Implementation and Development

Here is the main idea on how to implement and develop the code.

1. We were given the two audiofiles to extract the audio data vector and sample rate from.
2. Using the size of the audio data vector and sample rate, we would be able to calculate the total time of the audio clip. Along with this information, we are able to create our time and frequency domain.
3. The frequency domain is normally made like: $k = (2*\pi/2*L)*[0 : n/2-1-n/2 : -1]; ks = fftshift(k);$, however, FFT is in angular frequencies, but we are wanting to look at Hertz, which changes the constant $(2 * \pi/2 * L)$ to $1/L$.
4. We create a for loop that goes through each time in equally-spaced points. At each specific time, we multiply our audio data vector with the Gabor filter that is specifically made at this time, $t = \tau$.
5. We are able to choose our constant for the Gabor transform that allows us to get the best accuracy in both the frequency and time domain.
6. Afterwards, we take a Gaussian filter and filter the current vector around the maximum value, to decrease the amount of noise that will show up in our spectrogram.
7. We choose the constant for the Gaussian filter to make it as wide or as thin as we want it to filter.
8. Finally, we create the spectrogram and locate the notes by looking at what frequencies show up.

Algorithm 1: Creating the Fourier and Time Domain

Get audio data and sample rate from GNR.m4a and Floyd.m4a
Get the length of each clip by dividing the length of the audio data vector by the sample rate
Create the time domain which is just the length of the audio data
Create the frequency domain which is divided by L instead of 2π because we are dealing with Hz

Algorithm 2: Analyzing Audio Data Vectors

Choose a constant a for the Gabor Transform
Set up a vector for τ where all points are equally-spaced
for j = 1:length of vector τ **do**
  Create Gabor Transform filter
  Multiply with audio data vector
  Fourier Transform the resulting vector
  Find the strongest frequency at t=τ
  Create a Gaussian filter centered at the strongest frequency
  Multiply with the Fourier Transformed audio data vector
  Add the vector to a matrix that contains all the vectors at each $t = \tau$
end for
Graph the resulting matrix using pcolor
Find the hottest areas and note down the frequencies at certain times
Find the notes that correspond to those frequencies

4 Computational Results

Here are the results of recreating the Guitar Score for GNR, the Bass Score for Floyd, and the Guitar Score for Floyd.

4.1 The Guitar Score - GNR

This is the spectrogram we get from Gun N' Roses' *Sweet Child O' Mine*

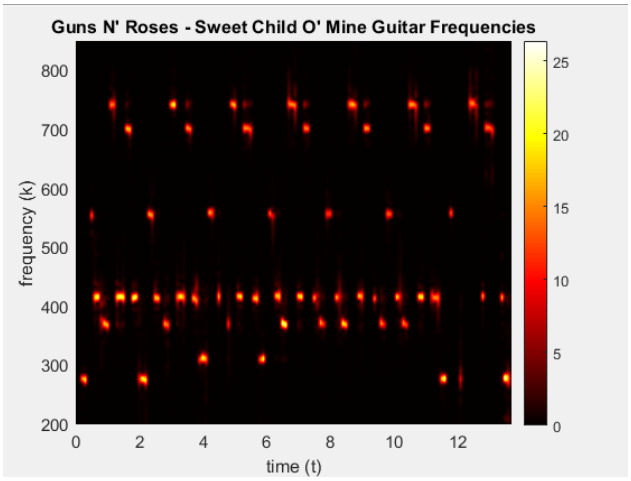

Figure 1: The spectrogram of the frequencies in GNR

With knowledge of what note corresponds to what frequency we can determine the first 8 notes to be:

Frequency (Hz)	Note
255	C#
547	C#
427	G#
370	F#
744	F#
410	G#
700	F
415	G#

Table 1: The first 8 notes to Sweet Child O' Mine

After these first 8 notes, the pattern repeats but with the first note being the only difference. The first two repeats are the same, having C# be the first note, however the next two repeats have the first note be a frequency of 298 Hz, making it a D. Then, the next two repeats have it at a frequency of 366 Hz, making it a F#. Finally, the last two repeats go back to the original frequency of 255 Hz, making it a C#

4.2 The Bass Score - Floyd

In this results section, I split up the first half and second half of Pink Floyd's analysis due to the fact it creates a large matrix that my computer fails to plot if it is whole.

This is the spectrogram of the first 30 seconds and second 30 seconds from Pink Floyd's *Comfortably Numb*

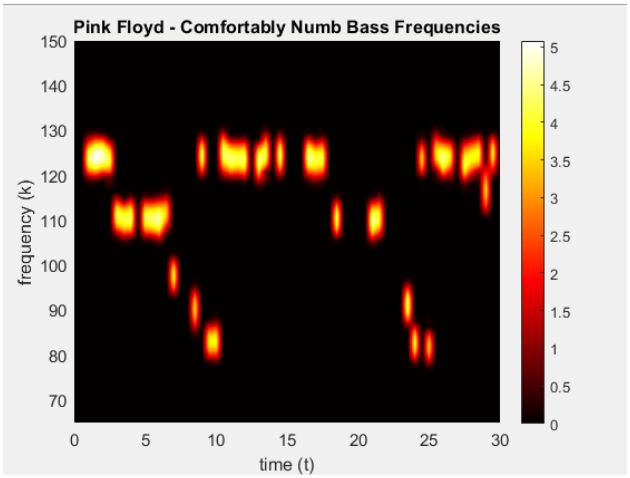

Figure 2: The spectrogram of the frequencies in the first half of Floyd

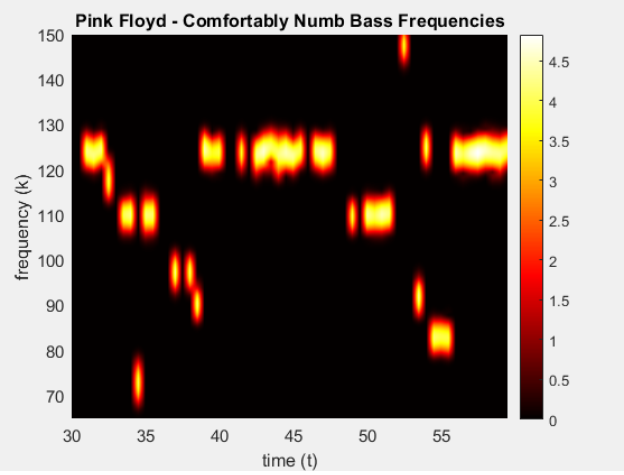

Figure 3: The spectrogram of the frequencies in the second half of Floyd

We notice that the notes drag on a lot more in this bass score than in the GNR section. Here is a graph showing which frequency is what note in the spectrogram:

Frequency (Hz)	Note
125	B
110	A
98	A
90	G
82	F#

Table 2: The Bass Frequencies and Corresponding Notes of Comfortably Numb

4.3 The Guitar Score - Floyd

Here I will show my best attempt to attain the Guitar Score for Pink Floyd's *Comfortably Numb*. I have also split up this clip into two spectrograms.

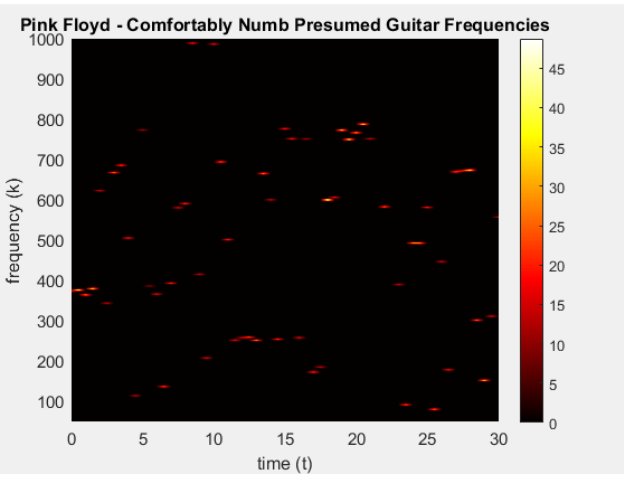

Figure 4: The spectrogram of the frequencies in the first half of Floyd

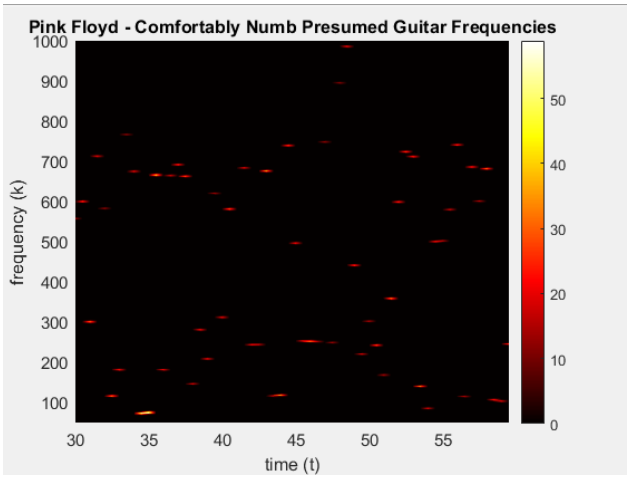

Figure 5: The spectrogram of the frequencies in the second half of Floyd

It's much harder clearly find the results for the guitar section, as there are some overtones from the bass showing up in this graph. With a bit more time, we should be able to polish this up and find a more accurate score of the guitar section.

5 Summary and Conclusions

With the use of the Gabor Transform, we were able to make these spectrograms as it gives us both a bit of freqnecy and time domain information. We are able to choose the constant that gives us the best mix of the two. Also, with the careful usage of the Gaussian Filter, we are able to clean our spectrograms to only show the hottest areas of the graph.

Appendix A MATLAB Functions

Add your important MATLAB functions here with a brief implementation explanation.

- `[y,Fs] = audioread(filename)` reads data from the file named filename, and returns sampled data, `y`, and a sample rate for that data, `Fs`.
- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `[row,col] = ind2sub(sz,ind)` returns the arrays `row` and `col` containing the equivalent row and column subscripts corresponding to the linear indices ind for a matrix of size `sz`. Here `sz` is a vector with two elements, where `sz(1)` specifies the number of rows and `sz(2)` specifies the number of columns.
- `Y = fftn(X)` returns the multidimensional Fourier transform of an N-D array using a fast Fourier transform algorithm. The N-D transform is equivalent to computing the 1-D transform along each dimension of `X`. The output `Y` is the same size as `X`.
- `Y = fftshift(X)` rearranges a Fourier transform `X` by shifting the zero-frequency component to the center of the array.
- `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

My MATLAB Code

```
%% Part 1 GNR Guitar Frequencies
clear; close all; clc;
[y1, Fs1] = audioread('GNR.m4a');
tr_gnr = length(y1)/Fs1; % record time in seconds

% Setting up Time and Frequency Domain
L = tr_gnr;
n = length((1:length(y1))/Fs1);
t2 = linspace(0,L,n+1); t = t2(1:n);
k = (1/L)*[0:n/2-1 -n/2:-1];
ks = fftshift(k);

% Turn y into a row vector
y = y1';

% Experimented with a to get a value that shows the frequencies well
a = 500; %
% Go across time using tau
tau = 0:0.1:L;
filt_Yft_spec = zeros(length(y), length(tau));
for j = 1:length(tau)
    % Create the Gabor Transform at t = tau
    filter = exp(-a*(t - tau(j)).^2);
    % Apply the filter
    Yf = filter .* y;
    % Go into Frequency space
    Yft = fft(Yf);

    % Find the max value in k
    [Max, Ind] = max(abs(Yft));
    [Max_Ind] = ind2sub(size(Yft), Ind);
    Max_Val = abs(k(Max_Ind));

    % Create the Gaussian Filter, centered around the max value of k in
    % t = tau
    fft_tau = 0.0001;
    fft_filt = exp(-fft_tau*(k - Max_Val).^2);
    filt_Yft = fft_filt .* Yft;

    filt_Yft_spec(:,j) = (fftshift(abs(filt_Yft)));
end

pcolor(tau, ks, abs(filt_Yft_spec));
shading interp
title("Guns N' Roses - Sweet Child O' Mine Guitar Frequencies")
% Set up the ylimit (which is frequency) to be between 200 and 850 Hz
% Because that is where the observed frequencies are
set(gca, 'ylim', [200 850], 'fontsize', 11)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (k)')
```

```
%% Part 2 - Floyd Bass Fequencies
clear; close all; clc;
% figure(2)
[y2, Fs2] = audioread('Floyd.m4a');
tr_floyd = length(y2)/Fs2;

% Create time and frequency domain
L = tr_floyd;
n = length((1:length(y2)-1)/Fs2);
t2 = linspace(0,L,n+1); t = t2(1:n);
k = (1/L)*[0:n/2-1 -n/2:-1];
ks = fftshift(k);

% y's length is initially odd and because we had to divide n by 2 in
% frequency domain, we have to lose the last column of data so the vectors
% are the same length.
y = y2(1:length(y2)-1)';

% Experimented with a to get a value that shows the frequencies well
a = 100;
% Go across time using tau
tau = 0:0.5:10;
filt_Yft_spec = zeros(length(y), length(tau));
floyd_ind = zeros(1,length(tau));
floyd_freq = zeros(1,length(tau));
for j = 1:length(tau)
    % Create the Gabor Filter
    gabor = exp(-a*(t - tau(j)).^2);
    Yf = gabor .* y;
    Yft = fft(Yf);

    % Find the maximum
    [Max, Ind] = max(abs(Yft));
    [Max_Ind] = ind2sub(size(Yft), Ind);
    Max_Val = abs(k(Max_Ind));

    % Create the Gaussian Filter with constant 0.1 and filter around the
    % maximum that we found earlier.
    fft_tau = 0.1;
    fft_filt = exp(-fft_tau*(k - Max_Val).^2);
    filt_Yft = fft_filt .* Yft;

    filt_Yft_spec(:,j) = (fftshift(abs(filt_Yft)));

end

pcolor(tau,ks,log(abs(filt_Yft_spec)+1));
shading interp
title("Pink Floyd - Comfortably Numb Bass Frequencies")
% Set up the ylimit (which is frequency) to be between 65 and 150 Hz
% Because we notice that is where all the bass lies in.
set(gca, 'ylim', [65 150], 'fontsize', 11)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (k)')
```

```
%% Part 3 - Floyd Guitar Frequencies
% clear; close all; clc;
% figure(2)
[y2, Fs2] = audioread('Floyd.m4a');
tr_floyd = length(y2)/Fs2;

L = tr_floyd;
n = length((1:length(y2)-1)/Fs2);
t2 = linspace(0,L,n+1); t = t2(1:n);
k = (1/L)*[0:n/2-1 -n/2:-1];
ks = fftshift(k);

% y's length is initially odd and because we had to divide n by 2 in
% frequency domain, we have to lose the last column of data so the vectors
% are the same length.
y = y2(1:length(y2)-1)';

pre_filt_Yft = fftshift(fft(y));
pre_filt_Yft(abs(pre_filt_Yft) >= 300) = 0;
pre_filt_Y = ifft(ifftshift(pre_filt_Yft));

% Experimented with a to get a value that shows the frequencies well
a = 100;
% Go across time using tau
tau = 30:0.5:L;
filt_Yft_spec = zeros(length(y), length(tau));
for j = 1:length(tau)
    % Create the Gabor Filter
    gabor = exp(-a*(t - tau(j)).^2);
    Yf = gabor .* pre_filt_Y;
    Yft = fft(Yf);

    % Find the maximum
    [Max, Ind] = max(abs(Yft));
    [Max_Ind] = ind2sub(size(Yft), Ind);
    Max_Val = abs(k(Max_Ind));

    % Create the Gaussian Filter with constant 0.1 and filter around the
    % maximum that we found earlier.
    fft_tau = 0.1;
    fft_filt = exp(-fft_tau*(k - Max_Val).^2);
    filt_Yft = fft_filt .* Yft;

    filt_Yft_spec(:,j) = (fftshift(abs(filt_Yft)));

end

pcolor(tau,ks,filt_Yft_spec);
shading interp
title("Pink Floyd - Comfortably Numb Presumed Guitar Frequencies")
% Set up the ylimit (which is frequency) to be between 0 and 150 Hz
% Because that is the range limit of a guitar
set(gca, 'ylim', [50 1000], 'fontsize', 11)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (k)')
```


Listing 1: Code for this homework.