AMATH 482 Assignment 2

Megan Freer

February 10, 2021

Abstract

Sounds all around us can be represented in terms of the frequencies of sound waves that produce those sounds. For example, music can be represented with the frequencies of various notes. Analyzing the audio data of famous rock and roll songs such as Sweet Child O' Mine by Guns N' Roses and Comfortably Numb by Pink Floyd allows us to reveal the notes being played by certain instruments. We can utilize Gabor transforms, the Fast Fourier Transform, and Gaussian filters in order to filter the data and find the notes being played over the course of the audio recordings.

1 Introduction and Overview

All sounds can be recorded as frequencies, with lower frequencies producing lower pitches and higher frequencies producing higher pitches. The audio signals of famous rock and roll songs, such as Sweet Child O' Mine by Guns N' Roses and Comfortably Numb by Pink Floyd, present an interesting case study for audio analysis. By importing and filtering this audio data, we are able to isolate particular instruments, such as a guitar or bass, and determine what notes each instrument is playing at each time point. This detection of instrument-specific notes is based on the specific frequency range of each instrument. For example, the bass is known to play sounds with frequencies 60 to 250 Hz, while the guitar is known to play sounds with frequencies above 250 Hz and up to approximately 1000 Hz.

In order to process this acoustic data, we will be using MATLAB to perform Gabor transforms, Fast Fourier Transforms (FFT), and Gaussian spectral filters in order to isolate the notes played by specific instruments, all of which are important tools in signal processing. For each time window, the audio data was used with a Gabor transform and then had a Fast Fourier Transform (FFT) performed on it. Then we found the center frequency of the music at each time window in order to filter around that frequency to get rid of overtones. This produces the de-noised audio to reveal a spectrogram of what frequencies (and their corresponding notes) were being played over the course of the audio recording.

2 Theoretical Background

The Gabor Transform, or the short-time Fourier transform (STFT), is an important mathematical tool that allows for filtering in the time domain. This time filter is slid across the time domain in order to consider all possible locations for the filter. Considering a filter function $g(t)$, then shifting by τ and multiplying by a function $f(t)$ gives the filtered function $f(t)g(t-\tau)$. In this equation, τ describes where the filter is centered at. The Gabor transform is then given by:

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

The function $\tilde{f}_g(\tau,k)$ gives information about the frequency components near time τ. The choice of filter $g(t)$ changes the results of the Gabor transform, which is why there is a subscript g on $\tilde{f}$. A Gaussian is one common choice of filter to use and is what we will be using in our analysis of our audio clips. When a Gaussian is chosen as the filter to be used, the inverse of the Gabor transform is given by:

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

When a Gaussian is used as the filter for implementation of a Gabor transform, we use the Gaussian spectral filter:

$$g(t-\tau) = e^{-a(t-\tau)^2} \tag{3}$$

There are two important parameters within this equation: a and τ. a represents the width of the window and must be greater than 0, while τ represents the center of the window. It is important to note that a small a corresponds with a wide window, and a large a corresponds with a thin window. If this window width is very large, we get the Fourier transform over the entire signal, meaning a large amount of frequency information, but no information about time. If the window is very small, we are not getting enough frequency information and are just looking at individual times along the signal. We must strike a balance between these two possibilities in order for the Gabor transform to return some information about time and some information about frequency.

If we want to use the Gabor transform on data, we need a discrete version. Considering a discrete set of frequencies:

$$k = m\omega_0 \tag{4}$$

$$\tau = nt_0 \tag{5}$$

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

$$\tilde{f}_g(m,n) = \int_{-\infty}^{\infty} f(t)g(t-nt_0)e^{2\pi i m\omega_0 t}dt \tag{6}$$

The implementation of the Gabor transform involves multiplying the signal by a (discrete) window function (or filter) and then applying the Fast Fourier Transform (FFT). Next, we move the window and repeat the process. In order to effectively convey the information given over the time windows, we use a type of graph called a spectrogram. The spectrogram involves stacking all of the Fourier transforms next to each other to make a plot and having the horizontal axis be the value of τ (window center) and the vertical axis be the frequency, with the color representing amplitude at that time window. This gives a still that visually represents the results. MATLAB has a built-in spectrogram function that is useful when representing the results of our analysis.

It is also important to note that in order to scale the frequency axis correctly in MATLAB, we scale our frequency array by 1/L, where L is the length of the data array being analyzed. Recalling from our knowledge of the equation for the Discrete Fourier Transform (and the accompanying Fast Fourier Transform), we only have a finite number of frequencies present: $k = 0, 1, 2..., N-1$. When implementing this transform, there is a possibility of aliasing, meaning that k and $k+N$ are the same in the DFT, and it is possible that a signal may be mistaken for a different signal. Due to this, we instead consider the frequencies given by:

$$k = -N/2, -N/2+1, ..., -1, 0, 1, 2, ..., N/2-1 \tag{7}$$

Once these frequencies are scaled by 1/L, the frequency array can be used in the rest of the analysis.

Overall, the Gabor transform can be used to recover information about a signal in both the time and frequency domain. This is due to 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. There is however a trade off between accuracy in time and frequency that must be balanced.

3 Algorithm Implementation and Development

Algorithms involving Gabor transforms, Fast Fourier Transforms and Gaussian filters were used to isolate the notes being played by the guitar in Sweet Child O' Mine by Guns N' Roses, the bass in Comfortably Numb by Pink Floyd, and the guitar in Comfortably Numb by Pink Floyd. Starting with the algorithm for the guitar in Sweet Child O' Mine by Guns N' Roses, the code starts by loading in the audio data, which comes in the form of a 659122x1 array representing the acoustic data of the song over the course of 13.7317 seconds. Reading this audio data also produces the sampling rate, which will be used later. Due to the large size of this data and the computational energy it takes the computer to run algorithms on such large arrays,

we then downsampled the audio data by a factor of 4 and readjusted the sampling rate down by a factor of 4, resulting in a 164781x1 audio array. This does not have a significant impact on the output of the entire algorithm, but it does make the code run significantly faster.

We then created an array of time points, as this will be used later for the Gabor transform. This time array goes from 1 over the sampling rate to 13.7317 seconds (the total length of the audio clip), with the same number of time increments as in the audio array. We also defined for later use the audio clip length, defined by the length of the audio data array over the sampling rate, and the number of samples, defined by the length of the audio data array.

We also defined a 1x164781 array of frequencies, as defined in Equation 7, due to aliasing. This array of frequencies is then re-scaled by 1 over the length of the audio clip in seconds in order to scale the axis correctly in MATLAB. This array of frequencies was then put into `fftshift` in order to shift the zero-frequency component to the center of the array. Next, we defined the variables that would be used in future Gabor transforms. The value of a was set to 100 (used for the Gabor transform filter) and the value of a for Gaussian filtering around a maximum frequency was set to 0.01 (used for filtering out overtones). We also defined the vector for τ, which ranges from 0 to 13.7317 seconds, with time steps equivalent to the the length of the audio clip over the number of distinct notes heard when listening to Sweet Child O' Mine all over five, which was the number of windows we wanted per note. We also at this point initialized the size of two matrices that will be used to store data later. The matrix storing the highest frequency in each time window is the length of τ by 1, and the matrix storing the spectrogram information is the length of the audio data by the length of τ.

The next step is to loop through each of the windows, or looping through τ. At each window, we create a filter as defined in Equation 3, with the a defined earlier. This filter is then element-wise multiplied by the transposed audio array (in order to match dimensions). The array that results then has a Fast Fourier Transform performed on it in order to transform the audio into the frequency domain. Next, at each time point (meaning at each iteration through the loop through τ), we find the maximum amplitude of the Gabor filtered audio array in the frequency domain, as well as the index of that maximum. In order to find the associated frequency, we find the value in the frequency array at that index. Next, we want to filter out frequencies above and below this maximum, particularly to filter out overtones. Overtones are the frequencies greater than the fundamental frequency that result from the physics of how instruments, such as guitars, work. For example, a lower note may show up at frequencies multiples of octaves higher than the actual note being played. In order to filter this out, we create another Gaussian filter, but this time in the frequency domain and centered around the center frequency that was previously found. In order to apply the filter, we multiplied the filter by the previously filtered audio signal in the frequency domain. We then performed an `fftshift` in order to get ready to plot, which shifts the zero-frequency component to the center of the spectrum, and saved this value in the matrix storing spectrogram data.

Outside of the loop through τ, we plotted a spectrogram with τ on the horizontal axis and the shifted frequencies on the vertical axis, with the color representing the log of the saved spectrogram data matrix plus one. The axes and spectrogram colors were adjusted to make the figure more clear. Next, we created a plot that is similar to the spectrogram, but appears more like a music score. This plot is a scatterplot of τ on the horizontal axis and the maximum frequency at each time window on the horizontal axis. This plot also includes note labels and horizontal lines representing the specific frequencies associated with certain notes. When a data point appears on one of these lines, we know that note is being played at that point in the audio clip.

This algorithm was repeated closely for finding the bass in Comfortably Numb by Pink Floyd, with a few modifications. Directly after reading in the audio data, we shortened the array to only include the first 30 seconds. This allowed for faster computing time and also clearer figures, as 60 seconds of varying notes would appear very messy on a chart. We also downsampled the data by 8x, rather than 4x as was done previously, in order to once again decrease computation time, this time especially with a longer audio clip. Another modification is that before entering the loop through the various time windows, we found the index location within the frequency array that corresponds to 60 Hz and 250 Hz. This is because these are the cutoff frequencies for the bass, so within the time window loop and before finding the maximum frequency, we set all values in the audio array corresponding to below 60 Hz or above 250 Hz to zero. This allowed us

to find the maximum of specifically the bass frequencies.

This same algorithm was once again repeated closely for isolating the guitar in Comfortably Numb by Pink Floyd, with a few more modifications. The data was shortened to only include the first 10 seconds of audio to analyze the faster note changes of the guitar solo. The audio array was once again downsampled by 8x in order to decrease computation time. Since the guitar includes frequencies greater than 250 Hz, we found the index within the frequency array where 250 Hz occurred. Then within the time window loop, we set all frequencies above 250 Hz (as well as all negative frequencies) to zero. This allowed us to find the maximum frequency of just the guitar, rather than the bass.

4 Computational Results

After performing these calculations, the MATLAB code was able to produce computational results for the frequencies being played by the instruments (guitar or bass) over the course of the audio clip length. The computational results for the guitar in Sweet Child O' Mine by Guns N' Roses can be see in Figure 1, with the spectrogram of filtered frequencies on the left, and a musical "score" with the notes associated with each frequency on the right. The general pattern of the guitar riff in this song is $D\flat - D\flat - A\flat - G\flat - G\flat - A\flat - F - A\flat$. These eight notes are repeated twice, then the first note of this eight-note pattern is replaced by $E\flat$ for two repeats through. Next, the first note of the eight-note pattern is replaced by $G\flat$ for two repeats through. Finally, the eight-note pattern returns to its original eight notes and starts with $D\flat$ once again. This corresponds with the key of Sweet Child O' Mine: D$\flat$ major, which includes $D\flat, E\flat, F, G\flat, A\flat, B\flat$, and C. Each of the notes appears to be at almost the exact frequency of a note in this key, which leads us to believe that the computational results match the notes that were played on the guitar.

The algorithm was also able find computational results for the bass in Comfortably Numb by Pink Floyd, meaning frequencies that were between 60 to 250 Hz. These results can be see in Figure 2, with the spectrogram of filtered frequencies on the left, and a musical "score" with the notes associated with each frequency on the right. Note that these figures only include the notes over the first 30 seconds of audio. Comfortably Numb is in B minor, which includes $B, C\#, D, E, F\#, G$, and A. Once again, the notes found for the bass in this song appear to match almost exactly the frequencies associated with the notes in this key. B is also a heavily repeated note by the bass, which aligns with the piece being in B minor and further verifies that the computational results are likely correct. Any differences between the computational results and the actual notes being played could be due to overtones that have a higher amplitude than the actual note in some time windows. This could lead to some maximum frequencies showing up an octave higher than the actual note being played, but this is difficult to eliminate since it is hard to tell which octave is correct at a certain time point besides simply using the maximum amplitude associated with the frequencies.

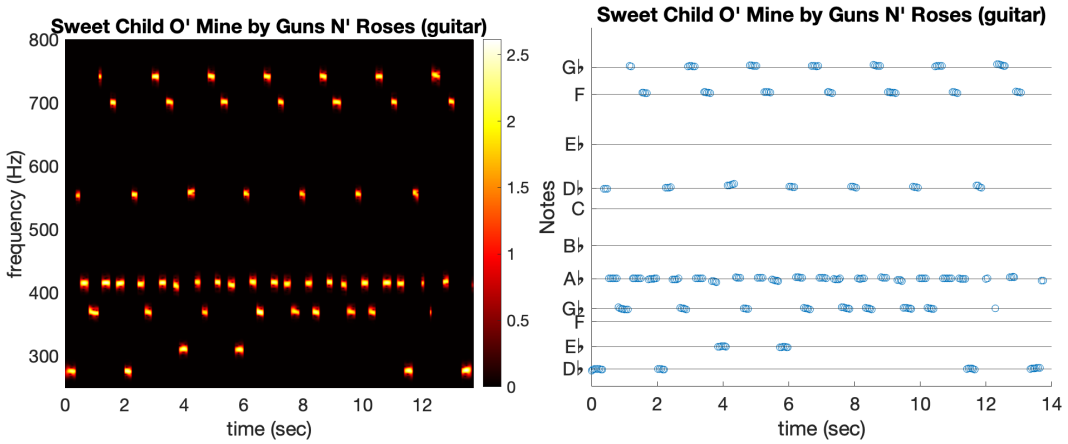

Figure 1: The frequencies and notes being played by the guitar in Sweet Child O' Mine by Guns N' Roses over the course of the first 14 seconds of the song. On the left is a spectrogram representing the frequencies at each time window and on the right is the music score with the notes being played at each time window.

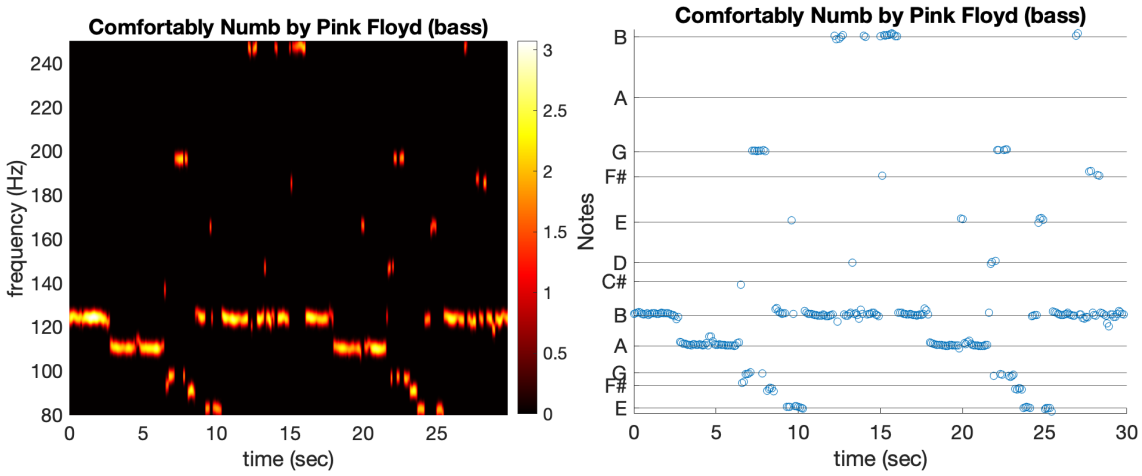

Figure 2: The frequencies and notes being played by the bass in Comfortably Numb by Pink Floyd over the course of the first 30 seconds of the audio clip. On the left is a spectrogram representing the frequencies at each time window and on the right is the music score with the notes being played at each time window.

We were also able to find the notes being played by the guitar in Comfortably Numb by Pink Floyd, as can be seen in Figure 3, with the spectrogram of filtered frequencies on the left and the musical "score" with the notes associated with each frequency on the right. The guitar plays notes between approximately 250 and 1000 Hz. Note that these figures only include the notes over the first 10 seconds of the guitar solo audio clip. Once again, these notes appear to line up relatively well with the notes in the key of B minor. There appears to be a few time windows where the maximum frequency does not correspond with the note being played, which could be due to overtones being dominant in that time window, as discussed earlier. However, these results do still appear to match the sound of the guitar very closely.

Overall, the algorithms used appear to give verifiable results for the notes being played by the bass and guitar in Sweet Child O' Mine and Comfortably Numb.

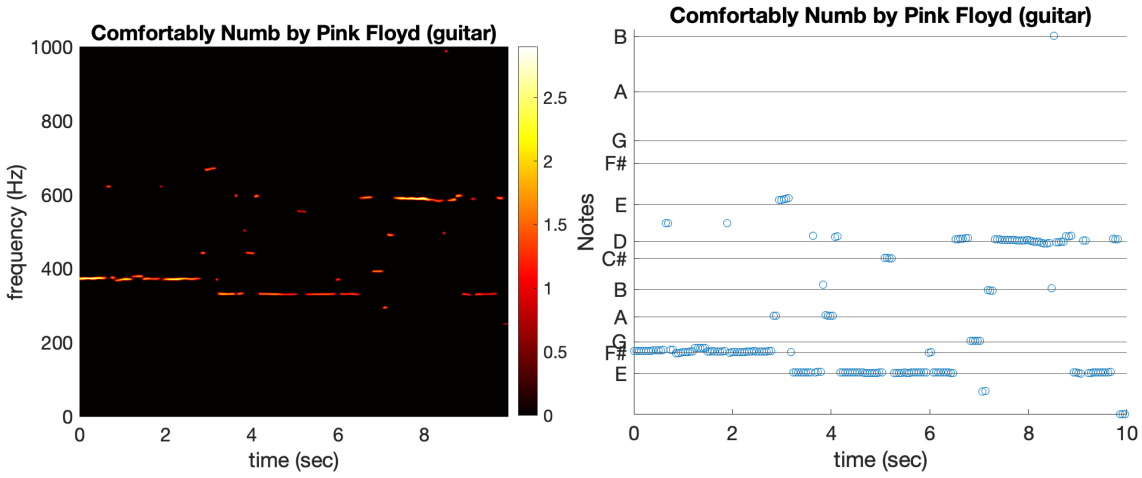

Figure 3: The frequencies and notes being played by the guitar in Comfortably Numb by Pink Floyd over the course of the first 10 seconds of the audio clip of the guitar solo. On the left is a spectrogram representing the frequencies at each time window and on the right is the music score with the notes being played at each time window.

5 Summary and Conclusions

In conclusion, Gabor transforms, Fast Fourier Transforms, and Gaussian filters are useful tools, particularly for analyzing audio data. These tools were able to find the notes being played by separate instruments (bass and guitar) for the songs Sweet Child O' Mine by Guns N' Roses and Comfortably Numb by Pink Floyd. When combined, these methods were able to filter out noise and overtones using MATLAB and provide a powerful tool for audio analysis. These important signal processing tools can be applied to other songs and audio, unlocking the possibility for audio analysis in many future applications.

Appendix A MATLAB Functions

The following MATLAB functions are implemented in this code:

- `[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 = downsample(x,n)` decreases the sampling rate of x by keeping the first sample and then every nth sample after the first. This allowed for faster computing times with the audio data.
- `length(obj)` returns the length of an object, such as the length of an array.
- `Y = fftshift(X)` rearranges a Fourier transform `X` by shifting the zero-frequency component to the center of the array. If `X` is a vector, then fftshift swaps the left and right halves of `X`.
- `X = zeros(n,m)` returns an n-by-m matrix of zeros.
- `Y=fft(X)` returns the discrete Fourier transform of X, meaning transforming the input matrix from the time domain to the frequency domain.
- `Y = abs(X)` returns the absolute value of each element in array X.
- `[max_value, max_index] = max(A)` returns the index into A that corresponds to the maximum value in the matrix A. This method also returns what this maximum value is.
- `pcolor(x,y,c)` creates a 2D plot of x and y, with c corresponding to the colors of each "cell".
- `shading interp` varies the color in each cell by interpolating the colormap index or true color value across the cell.
- `colormap(hot)` sets the colormap for the current figure to warm colors ranging from white/yellow to orange/red.
- `colorbar` displays a vertical colorbar to the right of the current chart that displays the current colormap and indicates the mapping of data values into the colormap.
- `scatter(x,y)` creates a scatter plot with circles at the locations specified by the vectors x and y.
- `k=find(x > n)` returns the indices in x that are greater than n.

Appendix B MATLAB Code

```
% Megan Freer
% AMATH 482
% Assignment 2

%% Part 1: Sweet Child O' Mine by Guns N' Roses (guitar)
% in D major
clear all; close all; clc;
```

```
% import and convert GNR (Sweet Child O' Mine)
[y, Fs] = audioread('GNR.m4a');

% Make computing faster (decreases sampling rate)
y = downsample(y,4);
Fs = Fs/4;

% Defining time vector, length of audio clip, and number of samples
t = (1:length(y))/Fs; % time points vector
clip_length = length(y)/Fs; % audio length in seconds
n = length(y); % number of samples

% Defining frequency vector and shifting zero-frequency component to center
% of spectrum
k = (1/clip_length)*[0:(n-1)/2 -(n/2):-1];
ks = fftshift(k);

% Gabor filtering (define window width and center of window tau)
a = 100; % used for Gabor filter
a_overtone = 0.01; % used for overtone filter (Gaussian filter)
num_notes = 57; % manually counted
windows_per_note = 5; % want multiple windows per note
tau = 0:(clip_length/num_notes)/windows_per_note:clip_length;

% Initializing matrices being stored
max_freq = zeros(length(tau), 1);
y_no_overtone_spec = zeros(length(y), length(tau));

for j = 1:length(tau)

    % Make Gabor Filter and apply to audio
    gabor_filter = exp(-a*(t - tau(j)).^2);
    y_gabor = gabor_filter.*y';

    % Transform audio to frequency domain
    y_gabor_freq = fft(y_gabor);

    % At each time point, find max amplitude and associated frequency
    [max_amp, index] = max(abs(y_gabor_freq));
    max_freq(j) = k(index);

    % Filter out overtone (in frequency domain)
    % Uses Gausian filter around center frequency
    overtone_filter = exp(-a_overtone*(k - k(index)).^2);
    no_overtone_freq = overtone_filter.*y_gabor_freq;

    % Getting ready to plot (shifting zero-frequency component to center of
    % spectrum and adding frequency data to spectrogram)
    y_no_overtone_spec(:,j) = fftshift(abs(no_overtone_freq));
end

% Plot spectrogram
figure()
```

```
pcolor(tau,ks,log(y_no_overtone_spec + 1))
shading interp; colormap(hot); colorbar;
ylim([250 800])
set(gca,'Fontsize',16)
xlabel('time (sec)'), ylabel('frequency (Hz)')
title("Sweet Child O' Mine by Guns N' Roses (guitar)");

% Plot "spectrogram" (music score) with max frequency at each timepoint
figure()
scatter(tau, max_freq)
ylim([250 800])
set(gca,'Fontsize',16)
xlabel('time (sec)'), ylabel('Notes')
title("Sweet Child O' Mine by Guns N' Roses (guitar)");
freq_plot = [277.18, 311.13, 349.23, 369.99, 415.3, 466.16, 523.25,...
             554.37, 622.25, 698.46, 739.99, 830.61];
yticks(freq_plot);
yticklabels({'D', 'E', 'F', 'G', 'A', 'B', 'C', 'D', 'E', 'F',...
             'G', 'A'});
for i = 1:length(freq_plot)
    yline(freq_plot(i));
end
%% Part 2: Comfortably Numb by Pink Floyd (bass)
% in B minor
clear all; close all; clc;

% import and convert Floyd (Comfortably Numb)
[y, Fs] = audioread('Floyd.m4a');

% Shorten to first 30 sec of audio clip for faster computing
y = y(1:length(y)/2);

% Make computing faster (decreases sampling rate)
y = downsample(y, 8);
Fs = Fs/8;

% Defining time vector, length of audio clip, and number of samples
t = (1:length(y))/Fs;  % time points vector
clip_length = length(y)/Fs; % audio length in seconds
n = length(y); % number of samples

% Defining frequency vector and shifting zero-frequency component to center
% of spectrum
k = (1/clip_length)*[0:(n-1)/2 -(n/2):-1];
ks = fftshift(k);

% Find the index of k for 60 Hz and 250 Hz (bass frequencies)
array_60_Hz = find(k > 60);
index_60_Hz = array_60_Hz(1);
array_250_Hz = find(k > 250);
index_250_Hz = array_250_Hz(1);

% Gabor filtering (define window width and center of window tau)
a = 100; % used for Gabor filter
```

```
a_overtone = 0.1; % used for overtone filter (Gaussian filter)
step_size = 0.1;
tau = 0:step_size:clip_length;

% Initializing matrices being stored
max_freq = zeros(length(tau), 1);
y_no_overtone_spec = zeros(length(y), length(tau));

for j = 1:length(tau)

    % Make Gabor Filter and apply to audio
    gabor_filter = exp(-a*(t - tau(j)).^2);
    y_gabor = gabor_filter.*y';

    % Transform audio to frequency domain
    y_gabor_freq = fft(y_gabor);

    % Filter to only include bass frequencies (60-250 Hz)
    y_gabor_freq(1:index_60_Hz - 1) = 0;
    y_gabor_freq(index_250_Hz + 1:length(y_gabor_freq)) = 0;

    % At each time point, find max amplitude and associated frequency
    [max_amp, index] = max(abs(y_gabor_freq));
    max_freq(j) = k(index);

    % Filter out overtone (in frequency domain)
    % Uses Gaussian filter around center frequency
    overtone_filter = exp(-a_overtone*(k - k(index)).^2);
    no_overtone_freq = overtone_filter.*y_gabor_freq;

    % Getting ready to plot (shifting zero-frequency component to center of
    % spectrum and adding frequency data to spectrogram)
    y_no_overtone_spec(:,j) = fftshift(abs(no_overtone_freq));

end

% Plot spectrogram
figure()
pcolor(tau,ks,log(y_no_overtone_spec + 1))
shading interp; colormap(hot); colorbar;
ylim([80 250])
set(gca,'Fontsize',16)
xlabel('time (sec)'), ylabel('frequency (Hz)')
title("Comfortably Numb by Pink Floyd (bass)");

% Plot "spectrogram" (music score) with max frequency at each timepoint
figure()
scatter(tau, max_freq)
ylim([80 250])
set(gca,'Fontsize',16)
xlabel('time (sec)'), ylabel('Notes')
title("Comfortably Numb by Pink Floyd (bass)");
freq_plot = [82.41, 92.5, 98, 110, 123.47, 138.59, 146.83, 164.81, 185,...
             196, 220, 246.94];
```

```
yticks(freq_plot);
yticklabels({'E', 'F#', 'G', 'A', 'B', 'C#', 'D', 'E', 'F#', 'G', 'A',...
             'B'});
for i = 1:length(freq_plot)
    yline(freq_plot(i));
end
%% Part 3: Comfortably Numb by Pink Floyd (guitar)
% in B minor
clear all; close all; clc;

% import and convert Floyd (Comfortably Numb)
[y, Fs] = audioread('Floyd.m4a');

% Shorten to first 10 sec of audio clip for faster computing
y = y(1:length(y)/6);

% Make computing faster (decreases sampling rate)
y = downsample(y, 8);
Fs = Fs/8;

% Defining time vector, length of audio clip, and number of samples
t = (1:length(y))/Fs; % time points vector
clip_length = length(y)/Fs; % audio total length in seconds
n = length(y); % number of samples

% Defining frequency vector and shifting zero-frequency component to center
% of spectrum
k = (1/clip_length)*[0:(n-1)/2 -(n/2):-1];
ks = fftshift(k);

% Find the index of k for 250 Hz (maximum bass frequency)
array_250_Hz = find(k > 250);
index_250_Hz = array_250_Hz(1);

% Gabor filtering (define window width and center of window tau)
a = 100; % used for Gabor filter
a_overtone_guit = 0.1; % used for guitar overtone filter (Gaussian filter)
step_size = clip_length/200;
tau = 0:step_size:clip_length;

% Initializing matrices being stored
max_freq_guitar = zeros(length(tau), 1);
y_no_overtone_spec = zeros(length(y), length(tau));

for j = 1:length(tau)

    % Make Gabor Filter and apply to audio
    gabor_filter = exp(-a*(t - tau(j)).^2);
    y_gabor = gabor_filter.*y';

    % Transform audio to frequency domain
    y_gabor_freq = fft(y_gabor);

    % Apply band pass filter to get rid of bass (get rid of below 250 Hz)
```

```
    no_bass_overtone_freq = y_gabor_freq;
    no_bass_overtone_freq(1:index_250_Hz) = 0;

    % Remove amplitudes of all negative frequencies
    no_bass_overtone_freq(length(y_gabor_freq)/2:length(y_gabor_freq)) = 0;

    % At each time point, find max amplitude and associated frequency of
    % guitar (center frequency)
    [max_amp_guitar, index_guitar] = max(abs(no_bass_overtone_freq));
    max_freq_guitar(j) = k(index_guitar);

    % Filter out overtones of guitar (in frequency domain)
    overtone_filter_guitar = exp(-a_overtone_guit*(k-k(index_guitar)).^2);
    y_guitar = overtone_filter_guitar.*no_bass_overtone_freq;

    % Getting ready to plot (shifting zero-frequency component to center of
    % spectrum and adding frequency data to spectrogram)
    y_no_overtone_spec(:,j) = fftshift(abs(y_guitar));

end

% Plot spectrogram
figure()
pcolor(tau,ks,log(y_no_overtone_spec + 1))
shading interp; colormap(hot); colorbar;
ylim([0 1000])
set(gca,'Fontsize',16)
xlabel('time (sec)'), ylabel('frequency (Hz)')
title("Comfortably Numb by Pink Floyd (guitar)");

% Plot "spectrogram" (music score) with max frequency at each timepoint
figure()
scatter(tau, max_freq_guitar)
ylim([250 1000])
set(gca,'Fontsize',16)
xlabel('time (sec)'), ylabel('Notes')
title("Comfortably Numb by Pink Floyd (guitar)");
freq_plot = [329.63, 369.99, 392, 440, 493.88, 554.37, 587.33, 659.25,...
             739.99, 783.99, 880, 987.77];
yticks(freq_plot);
yticklabels({'E', 'F#', 'G', 'A', 'B', 'C#', 'D', 'E', 'F#', 'G', 'A',...
             'B'});
for i = 1:length(freq_plot)
    yline(freq_plot(i));
end
```