# AMATH 482 Homework 2

lewisxy

February 9, 2021

**Abstract**

In this paper, we demonstrated the possibility of extracting music notes from music clips with Gabor Transform and spectrogram.

## 1 Introduction and Overview

In this paper, we presented a technique of extracting music notes from music clips. Unlike other signal analysis tasks where we only care about frequencies, music analysis requires us to know both the frequency (music notes) and when those frequency occurs. Such tasks cannot be done by existing techniques like Discrete Fourier Transform. The techniques described in this paper utilizes Gabor Transform, which allows adjustable time and frequency resolution for different scenarios. We will use the data generated by Gabor Transform to plot spectrograms to visually aid the analysis. We also developed an experimental technique for removing overtones in the music. We provided 2 real world examples (*Sweet Child O' Mine* by Guns N' Roses and *Comfortably Numb* by Pink Floyd) in this paper.

## 2 Theoretical Background

### 2.1 Discrete Gabor Transform

The core mathematical tool in our methods is Discrete Gabor Transform. Unlike Discrete Fourier Transform, which allows us to find the frequency composition of a discrete signal, Discrete Gabor Transform also provides some information in the the real (time) domain, which can be useful to perform time-frequency analysis. Mathematically, Discrete Gabor Transform can be represented as (1). $\omega_0, t_0$ are the corresponding frequency and time resolution, $m, n$ are integer values used to index frequency and time. $g$ is a symmetric real-valued windowing function satisfying (2).

$$\hat{f}_g(m,n) = \int_{-\infty}^{\infty} f(t)g(t - nt_0)e^{-2\pi i m\omega_0 t}dt \tag{1}$$

$$\int_{-\infty}^{\infty} |g(t)|^2 dt = 1 \tag{2}$$

In practice, we usually apply Gabor Transform by applying Fast Fourier Transform (FFT) on the product of the signal and a time dependent windowing function. For the windowing function, we generally choose Gaussian function (3).

$$g(t - nt_0) = e^{-a(t-nt_0)^2} \tag{3}$$

Note that we can multiply a constant to $g$ to satisfy the requirement (2), although that requirement is generally relaxed if we only want to plot spectrogram. Note that we will need to enforce this requirement if we need to restore the signal (with inverse Gabor Transform).

### 2.2 Fourier Uncertainty Principle

In general, a higher frequency resolution will require a longer signal (poorer time resolution), whereas a short signal (higher time resolution) will produce poorer frequency resolution. The fact that we cannot get high resolution in both time and frequency domains is commonly referred to **Fourier Uncertainty Principle**. For Gabor transform, we can use parameter $a$ to adjust this trade off.

### 2.3 Spectrogram

By computing the Gabor Transform at each discrete time window, we will have a set of frequency decompositions arranged by those time windows. We can plot a graph those color representing the intensity of a given frequency component in a given time window with time and frequency on each axis. Such graph is called a spectrogram, which enables time-frequency analysis.

### 2.4 Music Notes

Music notes are a set of symbols each representing a specific frequency. For convention, we use A4 to represent 440 Hz, and each consecutive note will be multiplied by a factor of $2^{1/12} \approx 1.059463$. Conveniently, on a typical piano, consecutive keys corresponding to consecutive notes. Therefore, we can use the piano key number as a way to quantify music notes. Since there is 88 keys on a piano and A4 is 49th key, let $f_n$ the frequency of $n$th key of a piano, we can write the following recurrance relationship between music notes,

$$f_{n+1} = 2^{1/12} f_n, \quad \text{for } n = 1, ..., 88, \quad f_{49} = 440 \tag{4}$$

and obtain the close form equation to compute the frequency for a given music note (piano key number),

$$f_n = 2^{(n-49)/12} \times 440, \quad \text{for } n = 1, ..., 88 \tag{5}$$

We can also compute the approximate piano key number with the following equation (essentially the inverse of the function above)

$$n = 12 \times \log_2(f/440) + 49, \quad \text{for } n = 1, ..., 88 \tag{6}$$

### 2.5 Overtones

While music instruments are typically tuned to produced sound at a specific frequency defined by its note (This frequency is called **fundamental frequency**). Depending on the acoustic features of the instrument (how it vibrates), it may produce secondary harmonics with frequencies of integer multiples of its fundamental frequency. This phenomenon is known as **overtones**. The exact distribution of the overtones can be used to characterized instruments. This is also why we can tell piano from violin even when they are playing the same music notes. From the musical perspective, each 12 notes can be grouped into a octave, and the frequency doubles for the same notes on consecutive octaves.

## 3 Algorithm Implementation and Development

In this section, we describes the different steps to analyze music notes in *Sweet Child O' Mine* by Guns N' Roses (referred as GNR clip) and *Comfortably Numb* by Pink Floyd (referred as Flyod clip). Implementations are provided in the form of MATLAB code in Appendix B, explanations of selected MATLAB functions are provided in Appendix A.

### Optional: Resampling

Typical music clips are represented as 1D signals sampled at a fixed sampling rate (i.e. 44100 sample/sec). To reduce computational load, we can resample the signal at a lower sampling rate. This can be done with `resample` command in MATLAB. Note that the amount of discrete frequencies we can extract depends on the number of samples, resampling with a sampling rate too low (i.e. close to or lower than the frequency of a music note) will impair our capability of extracting music notes.

### 3.1 Constructing Spectrogram

To construct the spectrogram, we first need to decide on a time window size $t_0$, and construct an array $\tau = nt_0$ representing the starting time (in seconds) of each window. Then, we construct a 2D array (dimensions are number of time windows and number of frequencies) to store the data for spectrogram. For each time window, we compute the windowing function $g$ using (3), multiply to the signal, and compute the Fast Fourier Transform (FFT) of the product. The result will be the frequency decompostions for this given time, and we stored it in the spectrogram data array. With all the data filled in, we plot this 2D array with color representation (using `pcolor` in MATLAB) as spectrogram. Note that we do NOT normalize frequency by $2\pi$ (we still normalize it by $L$, the length of the clip in seconds) as we are using real frequencies (with unit Hz or sec$^{-1}$). Due to Fourier uncertainty principle, we will need to adjust parameter $a$ for a good balance of time and frequency resolutions. Ideally, the resulted spectrogram will contain distinguishable bright dots representing specific music notes (defined by its frequency) being played at a given time.

### 3.2 Adjusting Spectrogram

With spectrograms, we are already able to tell certain frequencies are being played at given time. However, due to the exponential relationship of music notes and frequencies, it is not immediately obvious which note is being played. We can use the equation (5) to transform frequencies to music notes (piano key number). Note that we need to use absolute value of frequencies since FFT in MATLAB will give us wave numbers as $k = -N/2, ..., -1, 0, 1, ...N/2 - 1$. After adjusting the frequency scale, we can re-plot the spectrogram with frequencies linear to each music notes (MATLAB will do interpolation for us automatically). Ideally, we will find each bright dot lies exactly on the frequency of a music note, indicating a specific note being played at a given time.

### 3.3 Removing Overtones (experimental)

Sometimes, the spectrogram can be noisy due to overtones. We can attempt to remove them by assuming they have similar intensity patterns as the fundamental frequency on the spectrogram except different locations (integer multiples of its fundamental frequencies). Note that validity of this assumption can vary wildly depending on the instruments (and it can be completely wrong sometimes), and it's impossible to know without the knowledge of the underlying instruments. Based on this assumption, one way to remove overtones is to subtract the intensity of the note at fundamental frequencies from the note at overtone frequencies, and clip the spectrogram with minimum value of 0. However, since notes produced by the actual instruments may vary slightly from its fundamental, we will need to subtract an interval of frequencies. To make implementation easier, we take advantage of the exponential property of music notes (every 12 notes doubles the frequencies), and use -0.5 and +0.5 note from the designated note as the frequency interval, and linearly interpolate the values in between to accommodate sampling difference. Note that this method can only remove overtones that is a power of 2 (i.e. 2nd, 4th, 8th, ...) since other intervals cannot be easily represented as integer-valued difference of music notes (for example 3rd overtone has 3 times the frequency of fundamental, and it's 1.5 times the 2nd overtone, or around 7 notes above the note for the 2nd overtone). Due to limited time, the exact process may not be well documented here, please refer to MATLAB code for detail.

## 4 Computational Results

For GNR clip, we first down sample the data by a factor of 1/4. Then using $a = 10240$, $t_0 = 0.1$ sec, we plot the spectrogram (Figure 1a). We can clearly observe the bright dots as the music notes produced by the instruments. If we rescale the spectrogram using equation (5), and plot the spectrogram again (Figure 1b), we can observe that center for most bright spots lies on a line, indicating they are indeed music notes. If we zoom in a bit into the frequency range for guitar (Figure 1c), we can figure out notes being played (not in the order being played): 41, 43, 46, 48, 53, 57, 58, 59, which corresponding to C#4, D#4, F#4, G#4, C#5, F5, F#5, G5.

For Flyod clip, we first discard the data after 15 seconds (as the clip is too long, and the music seems to be repetitive to some extent), and down sample the data by a factor of 1/4. Then using $a = 5120$, $t_0 = 0.1$ sec, we plot the spectrogram (Figure 2a), and the re-scaled spectrogram (Figure 2b). Observe that the low frequency components (on the lower part of the re-scaled spectrogram) are quite blurry. This is because the scaling is log function, which stretches the lower frequencies and squeezes the higher frequencies. To extract the notes being played in this section (bass, frequency range 60 - 250 Hz), we recompute the spectrogram with $a = 160$ to gain more frequency resolution (Figure 2c). Because of the overtones, the only notes we are certain being played are those on the lower part of the graph: 20, 22, 23, 25, 27, which corresponding to E2, F#2, G2, A2, B2. Without knowing the specific acoustic property of the instrument being used, we cannot determine whether those notes on the upper part of the graph are overtones or not.

To extract the bass, we take the Fourier Transform of the signal and apply a box filter that only allows sound between 60 to 250 Hz to pass through, and transform back. Using this technique, we can isolate the bass, and the resulted music sounds reasonable.

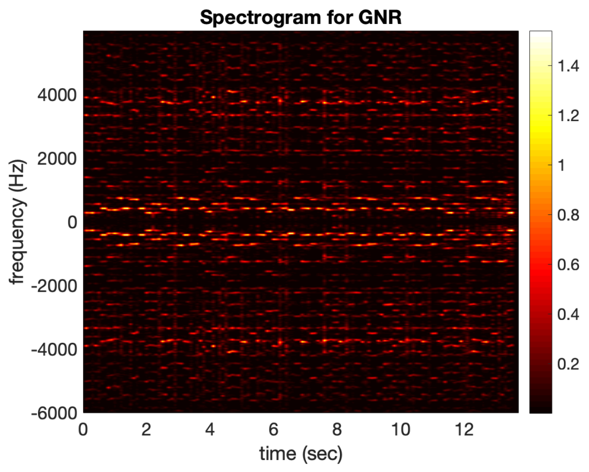
To extract the guitar from this clip, we first need to look into the frequency region for guitar (Figure 2d). Unfortunately, this region is noisy due to overtones from other instruments (primarily bass). To remove overtones, we use the method described in previous sections. The resulted spetrogram is indicated in Figure 2e. Due to imperfections of the music clip and overtone removal process, we can only identify the following notes from guitar: 42, 44, 46, 47, 54, or D4, E4, F#4, G4, D5. We should be able to utilize the inverse Gabor Transform to reproduce the audio after overtone (and bass) removal. Due to limited time, however, this is not implemented in this paper.

## 5 Summary and Conclusions

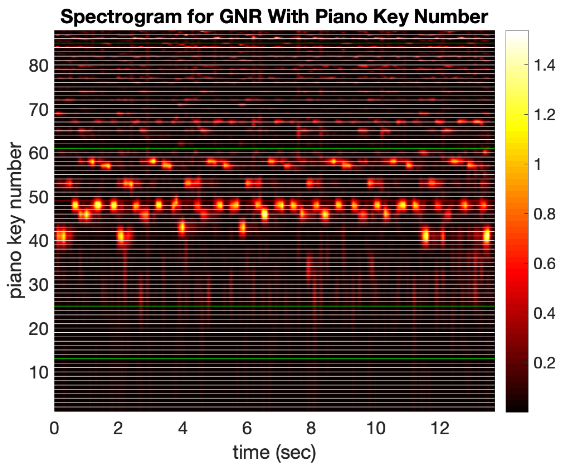
In this paper, we demonstrated the possibility of extracting music notes from music clips with Gabor Transform and spectrogram. These techniques enable us to perform time-frequency analysis efficiently, and has a wide range of applications. In the real worlds, due to imperfections of instruments and recordings, it can be difficult to extract the music notes accurately, especially when the music is composed of multiple instruments (with complex acoustic properties). In those cases, the techniques described in this paper can serve as a starting point to build more complex analysis tools to extract music notes.

## Appendix A MATLAB Functions

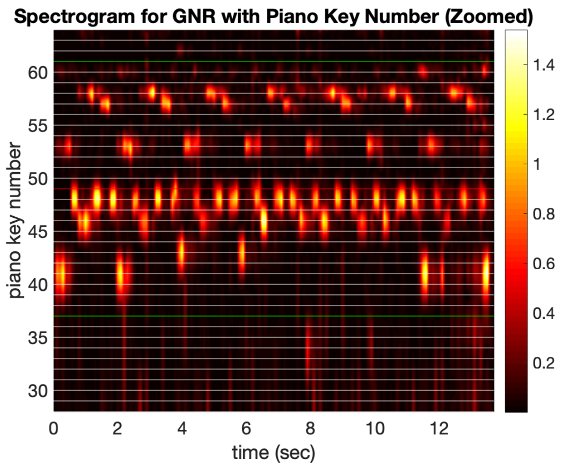
- `fft(x)`: perform Fast Fourier Transform (FFT) on the input.
- `fftshift(x)`: shift the output from `fft` in all dimensions so that frequencies are arranged in increasing order.
- `resample(x, p, q)`: re-sample the data given by `x`, for every `q` samples, produce `p` new samples.
- `pcolor(x, y, c)`: plot a 2D colored graph. `x`, `y` are 1D array defining input dimension and axis. `c` are 2D array with matching dimensions with `x` and `y` those value representing intensity of the color.



(a) Spectrogram for GNR Clip

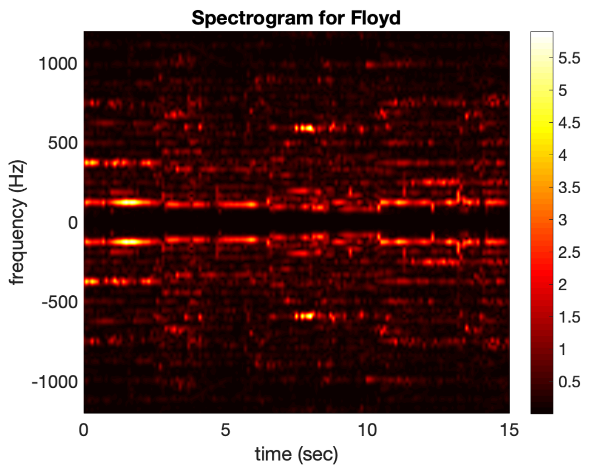


(b) Spectrogtram for GNR Clip, re-scaled with music notes, each white line represents a note, each green line represents A note to mark octaves, red line represents A4 = 440 Hz

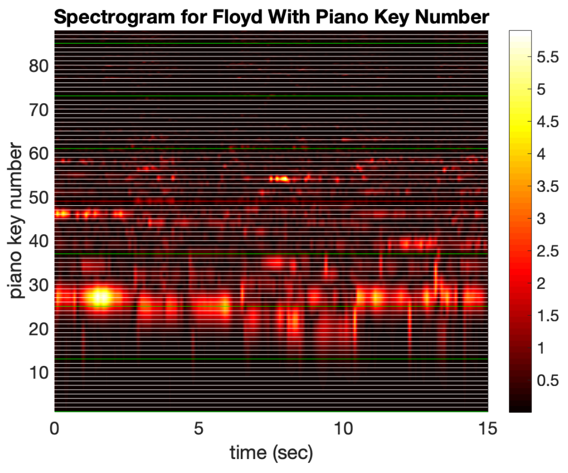


(c) Spectrogtram for GNR Clip, re-scaled with music notes for guitar, each white line represents a note, each green line represents A note to mark octaves, red line represents A4 = 440 Hz

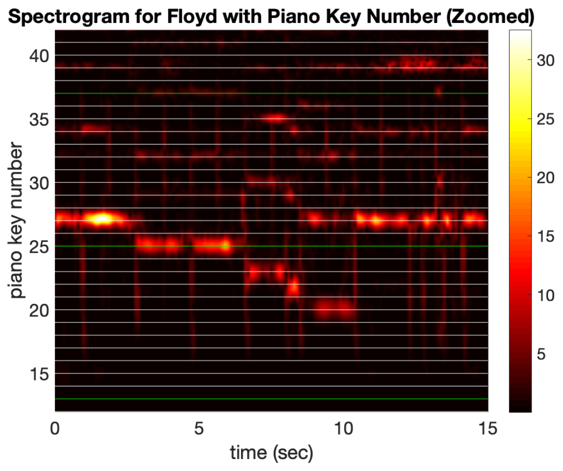
Figure 1: Analysis for GNR clips



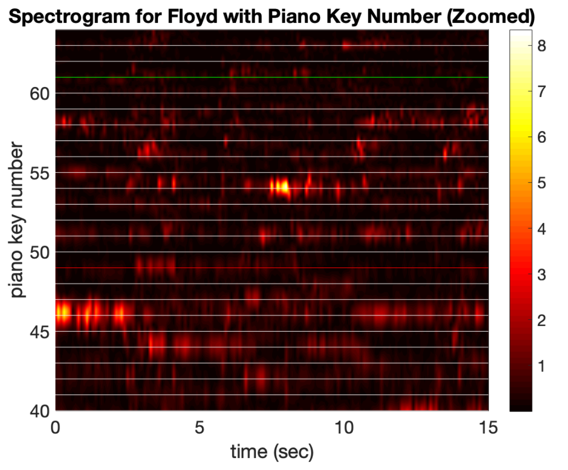
(a) Spectrogram for Floyd Clip



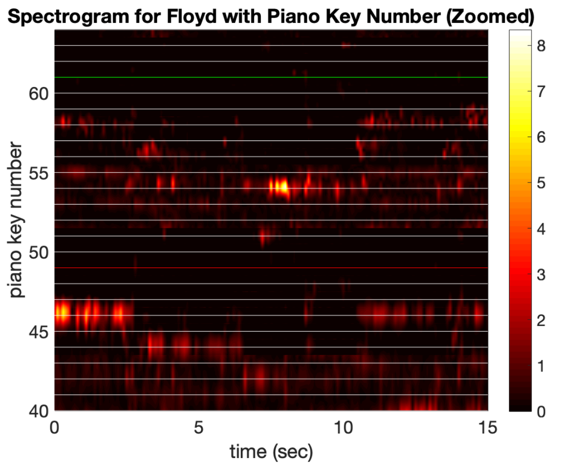
(b) Spectrogtram for Floyd Clip, re-scaled with music notes, each white line represents a note, each green line represents A note to mark octaves, red line represents A4 = 440 Hz



(c) Spectrogtram for Floyd Clip, re-scaled with music notes for bass, each green line represents A note to mark octaves



(d) Spectrogtram for Floyd Clip, re-scaled with music notes for guitar, each green line represents A note to mark octaves



(e) Spectrogtram for Floyd Clip, re-scaled with music notes for guitar after attempting to remove overtones from bass, each green line represents A note to mark octaves

Figure 2: Analysis for Floyd clips

## Appendix B MATLAB Code

```matlab
%% Cleanup workspace
clear all; close all; clc

%% Step 1: load and prepare data
[y, Fs] = audioread('data.nosync/GNR.m4a');
resample_factor = 1;

%% Option 1: downsample the data
% downsample data to make it easier to work with
% MATLAB will take a lot of time to plot spectrogram with original data
% and likely to produce memory error
resample_factor = 1/4; % 1 means original, < 1 means downsample
y = resample(y, resample_factor * Fs, Fs);
Fs = resample_factor * Fs;

%% Step 1 (continue): load constants
L = length(y) / Fs; % length of the song in sec
n = length(y); %  total number of samples
ts = (1:length(y)) / Fs; % time domain
if mod(n, 2) == 0
    k = (1/L)*[0:n/2-1 -n/2:-1]; % even length
else
    k = (1/L)*[0:n/2 -n/2:-1]; % odd length
end
ks = fftshift(k); % frequency domain

%% Step 2: create a spectrogram
% because of Fourier uncertainity, the less uncertainty in time means
% more uncertainty in frequency, we adjust parameter a to get a good
% balance of both

a = 10240;
tau = 0:0.1:ts(end);

s = y';
sgt_spec = zeros(length(ts), length(tau));
for j = 1:length(tau)
   g = exp(-a*(ts - tau(j)).^2); % Window function
   sg = g.*s;
   sgt = fft(sg);
   sgt_spec(:, j) = fftshift(abs(sgt));
end

%% Step 2 (continue): plot spectrogram
figure()
pcolor(tau, ks, sgt_spec)
shading interp
set(gca,'ylim', [ks(1) ks(end)], 'Fontsize', 16)
colormap(hot)
colorbar
xlabel('time (sec)'), ylabel('frequency (Hz)')
title('Spectrogram for GNR');
```

```
%% Step 3: adjust spectrogram to figure out musical note
% convert frequency to piano key number
% https://en.wikipedia.org/wiki/Piano_key_frequencies
freq2music = @(x) (12 .* log2(x / 440) + 49);
log_ks = freq2music(abs(ks));

% plot spectrogram with frequency axis rescaled to piano key number
figure()
pcolor(tau, log_ks, sgt_spec) % ks
shading interp
set(gca,'ylim', [1 88], 'Fontsize', 16)
colormap(hot)
colorbar
xlabel('time (sec)'), ylabel('piano key number')
title('Spectrogram for GNR With Piano Key Number');

% plot lines to indicate piano key
hold on;
for i=1:88
    line([0 L], [i i], 'Color', 'white')
    if mod(i, 12) == 1 % highlight every octave
        line([0 L], [i i], 'Color', 'green')
    end
end
line([0 L], [49 49], 'Color', 'red') % highlight key 49 at 440 Hz

%% plot again with zoomed in view
lb = 28; ub = 64;

figure()
pcolor(tau, log_ks, sgt_spec) % ks
shading interp
set(gca,'ylim', [lb ub], 'Fontsize', 16)
colormap(hot)
colorbar
xlabel('time (sec)'), ylabel('piano key number')
title('Spectrogram for GNR with Piano Key Number (Zoomed)');

% plot lines to indicate piano key
hold on;
for i=lb:ub
    line([0 L], [i i], 'Color', 'white')
    if mod(i, 12) == 1 % highlight every octave
        line([0 L], [i i], 'Color', 'green')
    end
end
line([0 L], [49 49], 'Color', 'red') % highlight key 49 at 440 Hz

% Music note being played are:
% 41 43 46 48 53 57 58 59
```

Listing 1: Complete MATLAB code for GNR clip

```
%% Cleanup workspace
clear all; close all; clc

%% Step 1: load and prepare data
[y, Fs] = audioread('data.nosync/Floyd.m4a');
resample_factor = 1;

%% Option 1: truncate the data
% the entire audio sample is large, we truncate it to
% avoid memory error
y = y(1:(Fs * 15)); % sample first 15 seconds

%% Option 2: downsample the data
% downsample data to make it easier to work with
% MATLAB will take a lot of time to plot spectrogram with original data
% and likely to produce memory error
resample_factor = 1/4; % 1 means original, < 1 means downsample
y = resample(y, resample_factor * Fs, Fs);
Fs = resample_factor * Fs;

%% Step 1 (continue): load constants
L = length(y) / Fs; % length of the song in sec
n = length(y); %  total number of samples
ts = (1:length(y)) / Fs; % time domain
if mod(n, 2) == 0
    k = (1/L)*[0:n/2-1 -n/2:-1]; % even length
else
    k = (1/L)*[0:n/2 -n/2:-1]; % odd length
end
ks = fftshift(k); % frequency domain

% convert frequency to piano key number
% https://en.wikipedia.org/wiki/Piano_key_frequencies
freq2music = @(x) (12 .* log2(x / 440) + 49);
music2freq = @(x) (2.^((x - 49) / 12) .* 440);
log_ks = freq2music(abs(ks));

%% Step 2: create a spectrogram
% because of Fourier uncertainity, the less uncertainty in time means
% more uncertainty in frequency, we adjust parameter a to get a good
% balance of both

% to make the result consistent, we adjust a based on resample_factor
%a = 5120;
%a = 160; % high freq resolution
a = 2560;
tau = 0:0.1:ts(end);

s = y';
sgt_spec = zeros(length(ts), length(tau));
for j = 1:length(tau)
   g = exp(-a*(ts - tau(j)).^2); % Window function
   sg = g.*s;
   sgt = fft(sg);
   sgt_spec(:, j) = fftshift(abs(sgt));
end

%% Step 2 (continue): plot spectrogram
figure()
pcolor(tau, ks, sgt_spec)
shading interp
set(gca,'ylim', [max(ks(1), -1200) min(ks(end), 1200)], 'Fontsize', 16)
colormap(hot)
colorbar
xlabel('time (sec)'), ylabel('frequency (Hz)')
title('Spectrogram for Floyd');

%% Step 3: adjust spectrogram to figure out musical note
% plot spectrogram with frequency axis rescaled to piano key number
figure()
pcolor(tau, log_ks, sgt_spec) % ks
shading interp
set(gca,'ylim', [1 88], 'Fontsize', 16)
colormap(hot)
colorbar
xlabel('time (sec)'), ylabel('piano key number')
title('Spectrogram for Floyd With Piano Key Number');

% plot lines to indicate piano key
hold on;
for i=1:88
    line([0 L], [i i], 'Color', 'white')
    if mod(i, 12) == 1 % highlight every octave
        line([0 L], [i i], 'Color', 'green')
    end
end
line([0 L], [49 49], 'Color', 'red') % highlight key 49 at 440 Hz

%% plot again with zoomed in view (bass)
lb = 12; ub = 42; % roughly 60 to 250 Hz

figure()
pcolor(tau, log_ks, sgt_spec) % ks
shading interp
set(gca,'ylim', [lb ub], 'Fontsize', 16)
colormap(hot)
colorbar
xlabel('time (sec)'), ylabel('piano key number')
title('Spectrogram for Floyd with Piano Key Number (Zoomed)');

% plot lines to indicate piano key
hold on;
for i=lb:ub
    line([0 L], [i i], 'Color', 'white')
    if mod(i, 12) == 1 % highlight every octave
        line([0 L], [i i], 'Color', 'green')
    end
end
```

```
% Music note being played are:
% 27 25 23 22 20 34 35 39
% after removing overtones
% 27 25 23 22 20

%% filter bass
% bass typically has a range of 60 to 250 Hz
box_filter = abs(ks) <= 250 & abs(ks) >= 60;
% apply filter
st = fftshift(fft(s));
sft = st.*box_filter;
sf = ifft(ifftshift(sft));

% sf contains the audio for isolated bass
% we can play it (by uncommenting the next line)
%p8 = audioplayer(sf, Fs); playblocking(p8);

%% plot again with zoomed in view (guitar)
lb = 40; ub = 64;

figure()
pcolor(tau, log_ks, sgt_spec) % ks
shading interp
set(gca,'ylim', [lb ub], 'Fontsize', 16)
colormap(hot)
colorbar
xlabel('time (sec)'), ylabel('piano key number')
title('Spectrogram for Floyd with Piano Key Number (Zoomed)');

% plot lines to indicate piano key
hold on;
for i=lb:ub
    line([0 L], [i i], 'Color', 'white')
    if mod(i, 12) == 1 % highlight every octave
        line([0 L], [i i], 'Color', 'green')
    end
end
line([0 L], [49 49], 'Color', 'red')

% Music note being played are:
% 27 25 23 22 20 34 35 39

%% removing overtones (experimental)
% try to subtract the bass frequency region from the guitar region

% the range of notes we want to remove
bass_notes = 20:27;

out = zeros(size(sgt_spec));
for j = 1:length(tau)
    % copy spectrogram
    out(:, j) = sgt_spec(:, j);
    for i=1:length(bass_notes)
```

```
        % compute note interval
        note_lb = bass_notes(i) - 0.5;
        note_ub = bass_notes(i) + 0.5;
        % compute corresponding frequencies
        mask_note = log_ks' > note_lb & log_ks' < note_ub;
        res = mask_note .* sgt_spec(:, j);
        count_note = sum(mask_note);
        notes = nonzeros(res);
        % remove up to 3 overtones (2nd, 4th, 8th)
        for k=1:3
            % compute the destination note and frequency range
            overtone_note_lb = note_lb + 12 * k;
            overtone_note_ub = note_ub + 12 * k;
            mask_overtone = log_ks' > overtone_note_lb & log_ks' < overtone_note_ub;
            res = mask_overtone .* sgt_spec(:, j);
            count_overtone = sum(mask_overtone);
            % interpolate values so that it fit the new range
            overtone_notes = resample(notes, count_overtone, count_note);
            % apply subtraction, clip to 0
            out(mask_overtone, j) = max(0, out(mask_overtone, j) - overtone_notes);
        end
    end
end

%% plot again with zoomed in view (guitar) again after removing overtones
lb = 40; ub = 64;

figure()
pcolor(tau, log_ks, out) % ks
shading interp
set(gca,'ylim', [lb ub], 'Fontsize', 16)
colormap(hot)
colorbar
xlabel('time (sec)'), ylabel('piano key number')
title('Spectrogram for Floyd with Piano Key Number (Zoomed)');

% plot lines to indicate piano key
hold on;
for i=lb:ub
    line([0 L], [i i], 'Color', 'white')
    if mod(i, 12) == 1 % highlight every octave
        line([0 L], [i i], 'Color', 'green')
    end
end
line([0 L], [49 49], 'Color', 'red')

% Music note being played are:
% 42 44 46 47 54
```

Listing 2: Complete MATLAB code for Floyd clip