AMATH 482 Homework 2: Audio Detection and Note Extraction

Amelia Nathan

February 10, 2021

Abstract

This paper develops hypothetical foundations of time-frequency signal processing and extends the Fourier Transform to the Gabor Transform to process audio files and detect the notes being played that make up the music file. Using the Gabor Transform and filtering out frequencies to isolate bass and guitar as desired, we are able to successfully use spectrograms to represent the music while overcoming limitations inherent in time-frequency analysis stemming from Heisenberg's Uncertainty Principle.

1 Introduction and Overview

1.1 Introduction to Signal Processing

Time-Frequency signal processing is a versatile area of study with extensive applications including seismic waves, echolocation, and vocals [3]. As will be seen in this paper as an example, processing audio files has become increasingly valuable in tasks like speech recognition and transcription. However, as signal processing has been used a variety of fields, limitations have also arisen; attaining localized information about both frequency and time is difficult. As will be discussed, the Heisenberg Uncertainty principle highlights this characteristic issue in signal processing applications [3]. It is therefore an ongoing challenge to best analyze data, especially since time variable data like music recordings are relevant to many investigations [3].

1.2 Problem Overview

This paper highlights the extensive application areas and potential of time-frequency processing by focusing on a specific example: reproducing music notes from an audio recording. Presented with 2 audio files of music, one 14 and one 59 seconds long, we are tasked with developing the music score for the guitar portion of the 14 second clip of Guns N' Roses' Sweet Child O' Mine and bass and guitar parts of Pink Floyd's Comfortably Numb.

2 Theoretical Background

Spectral transforms form a fundamental component of solving problems in applied mathematics; Fourier transforms more specifically, for example, enable representing functions as series of sines and cosines, making data visualization more clear [2]. While there are variations on the Fourier Transform, they are all based on this foundational formula that utilizes integration over an infinite domain [2]:

$$\hat{f}(k) = \frac{1}{\sqrt{2 * \pi}} * \int_{-\infty}^{\infty} f(x) * e^{-ikx} dx \tag{1}$$

To reverse the effects of the Fourier Transform, we can use the inverse:

$$f(x) = \frac{1}{\sqrt{2 * \pi}} * \int_{-\infty}^{\infty} \hat{f}(k) * e^{-ikx} dx \tag{2}$$

As previously learnt, a key theoretical characteristic of the Fourier Transform relates to the Heisenberg Uncertainty Principle. Founded in quantum mechanics, this principle emphasizes the limitations in attaining exact information about a particle's momentum and position; similarly, in performing Fourier Transforms we cannot know both the exact time and frequency of a signal [2]. There is a trade off in time-frequency analysis between localizing information in time and achieving accurate frequency information, meaning carefully chosen time windows and frequency spectrum are essential [2]. Ultimately, Fourier Transforms are well-suited to stationary signals that, when averaged, yield the desired signal [2]. This implication of the Uncertainty Principle poses a significant barrier in non-stationary signal analysis where the average Fourier components change in time–for example, as seen in songs and music.

To overcome this limitation, applying a Fourier Transform to a specified window in time proved to be an effective method to retain time and frequency information. Windowed Fourier Transforms, more specifically, the Gabor Transform, can therefore be used in time-relevant applications like speech and song analysis. Gabor Denes codified this idea in the Gabor Transform, based on the addition of $g(\tau - t)$ to the core of the Fourier Transform, which creates a time filter. Formally, the Gabor Transform is:

$$\tilde{f}_g(t,k) = \int_{-\infty}^{\infty} f(t)g(t-\tau)e^{-ikt}\,dt \tag{3}$$

By integrating, the time window slides over the entire signal, allowing analysis of the whole domain while retaining information about time. Equation 3 is typically based on some assumptions.

1. g is real and symmetric
2. $||g||_2 := (\int_{-\infty}^{\infty} |g(t)|^2 dt)^2 = 1$, meaning the L-2 Norm of f is set to unity [2].

As with the Fourier Transform, the Gabor Transform has an inverse and is defined by:

$$f(t) = \frac{1}{\sqrt{2}\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \tilde{f}_g(\tau,k)g(t-\tau)e^{ikt}dk d\tau \tag{4}$$

Also like the Fourier Transform, a discrete Gabor Transform allows us to apply it to recorded data by adapting it for discrete information:

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

In this equation $k = m\omega_0$ and $\tau = nt_0$, defining discrete frequencies and limiting tau to depend on those frequencies. ω_0, t_0 are positive constants defining the frequency resolutions.
As with the Fourier Transform, filter functions, such as the Gaussian below, are used as filter functions and are denoted by g in the equations above. We will use a Gaussian filter in this investigation.

$$g(t-\tau) = e^{-a(t-\tau)^2} \tag{6}$$

In practice and as will be seen in the implementation section, we must include a variable a in addition to τ to facilitate using a windowed transform. τ determines the centered location of the filter, which moves in discrete increments over the entire signal domain. a fixes the width of the window. Again, the Heinsenberg Uncertainty Principle comes into play as we seek the best values for a and tau. Smaller windows can provide more accurate time information, but they also exclude more frequency information. Choosing these values also presents some practical computation challenges, as will be discussed in the implementation section.

Finally, a foundational idea used in this investigation is that of spectrograms. When tasked with data where both time and frequency is important, presenting information can be difficult as visuals cannot easily utilize moving parts to represent, for example, the sliding window in analyzing an audio recording. Therefore, we can use a spectrogram that maps τ horizontally and frequencies vertically. Color saturation indicates the proportional presence of frequencies at different points in the domain.

3 Algorithm Implementation and Development

3.1 Implementation and Computational Process

We begin by using `audioread()` and `plot` to process and display our initial raw data of the audio files. For each song, we then:

1. Set L as the length of the song
2. Create a vector with equally spaced sections for the signal data
3. Rescale the frequency to fit our spacial domain and convert to hertz; rather than using 2π like in the previous assignment, dividing by L as seen in the appendix allowed us to successfully rescale to hertz.
4. Use `fftshift` to recenter frequencies

After these initial steps, we move on to using a Gaussian windowed function to process our data as seen in the first algorithm.

Algorithm 1: Processing Audio Signal

```
Define a, tau as the window width and discrete shifts
for every time step do
    Create a Gaussian filter
    Use Gaussian to filter signal
    Use fft to transform data into frequency domain
    Add to matrix
end for
Plot Spectrogram
```

When filtering for guitar or bass specifically, we add a rectangular filter to limit the range of data we want to look at.

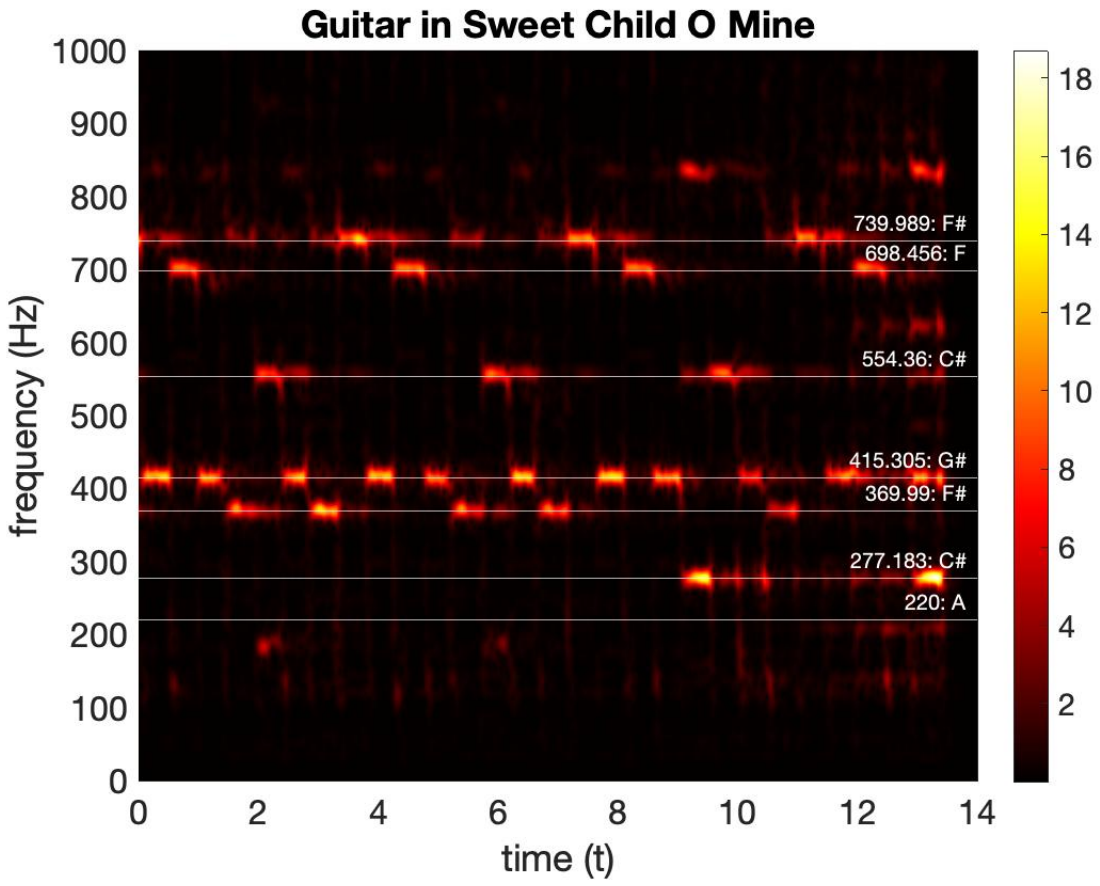
It is important to note that for the Pink Floyd song I split the raw data from the initial file import into two subsections since processing the entire file at once was not working on my computer.

3.2 Challenges and Limitations

This project highlighted the importance of computational power and efficient coding. Even after splitting the Floyd song into parts, my computer still took a long time to run and froze about half of the time. This significantly slowed down the process and limited my ability to test a wide range of a and tau values.

4 Computational Results

In the following spectrograms are my results from filtering the signals as described above. The Guitar in "Sweet Child O' Mine" was most clear in terms of filtering and isolating the notes. I added horizontal lines to the spectrogram with note labels to give an idea of the what specific notes are being played. While they do not all line up, they roughly map out some of the lines as would be seen on sheet music. I considered including all note lines to fully resemble sheet music, but this crowded the spectrogram so I opted to include a variety of notes near the frequencies detected.



As seen in the spectrogram, I was able to detect repeated notes around 415 and 369 Hz, corresponding yo G and F. There are also clear notes at the C corresponding to 554.36 Hz and F and F at 698.456 and 739.989Hz. The next page maps results for the Floyd song, which was much more difficult to pull exact notes out of. I was able to isolate the bass between about 70 and 150, although some harmonics remained. There is, however, clearly some B notes being played, corresponding to 123.47 Hz. In the guitar portion I am able to identify repeated Bs and Fs.

Because of how long each run through each part of the code took, I had difficult tracking improvement based on different a and tau values, let alone testing sufficient possible values for the clearest spectrogram. Were more computing power and time available, I believe attaining a better guitar score for Floyd is possible. I would like to try implement methods described in "Harmonic Decomposition of Signals With Matching Pursuit," which extends the Gabor transform method used in algorithm 1 to identify and eliminate harmonics and better focus the spectrogram [1].

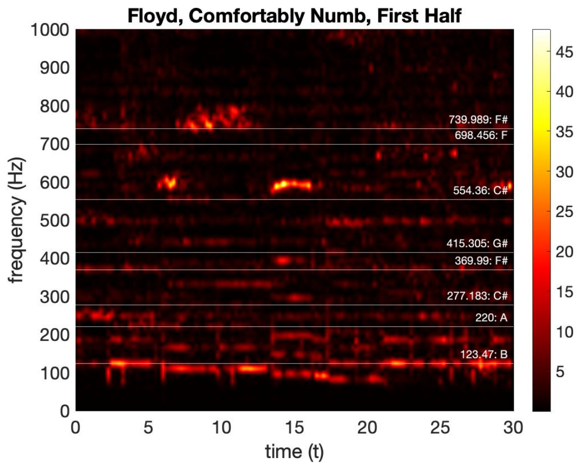


Figure 1: First Half Unfiltered

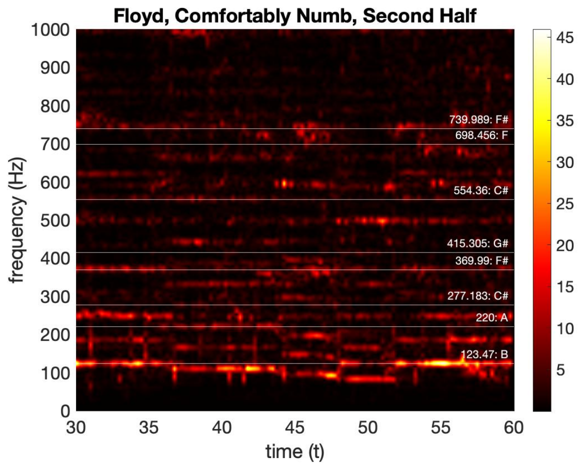


Figure 2: Second Half Unfiltered

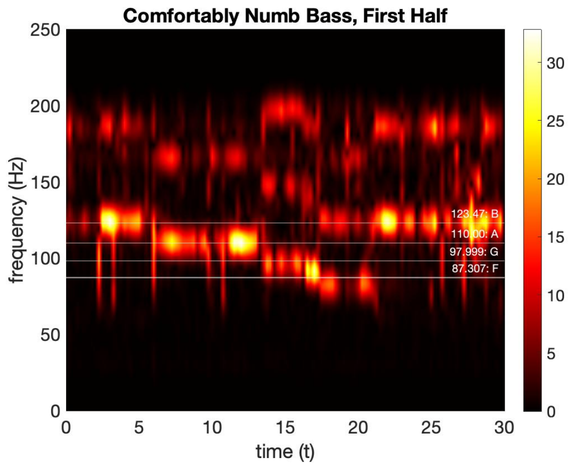


Figure 3: Isolated Bass

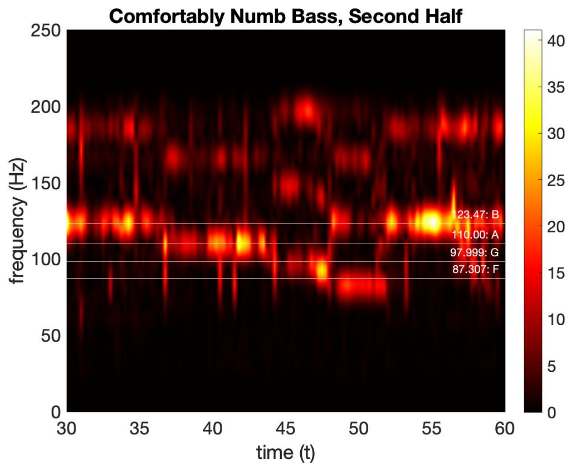


Figure 4: Isolated Bass

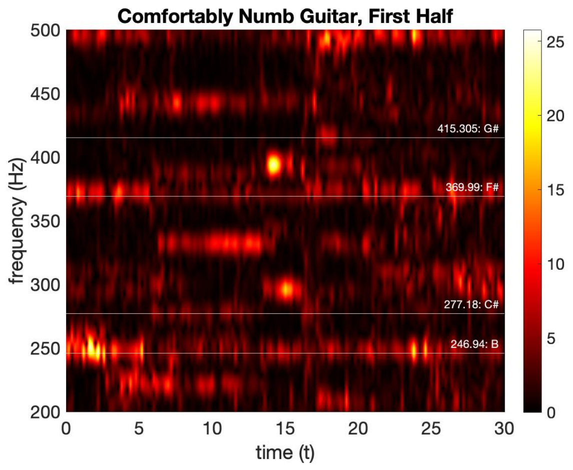


Figure 5: Isolated Guitar

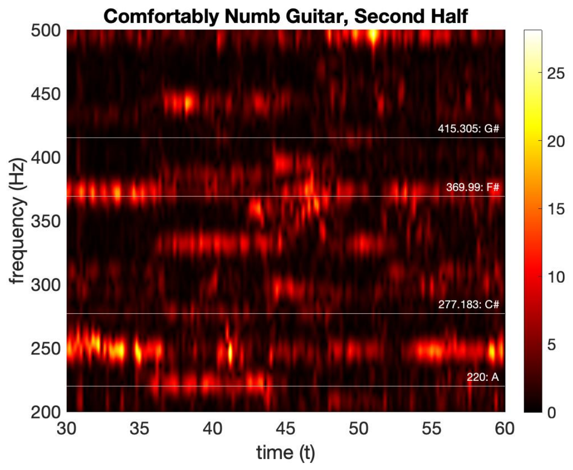


Figure 6: Isolated Guitar

5 Summary and Conclusions

While this investigation was challenging as processing data and developing spectrograms took a very long time for each run, I was ultimately able to identify some notes in both songs. The Floyd song, due to the presence of several instruments, was more difficult. However, I was able to estimate the ranges for bass and guitar portions of the Floyd song and apply appropriate rectangular filters to focus on those ranges and better help me see music notes in the spectrogram.

References

[1] Remi Gribonval and Emmanuel Bacry. "Harmonic Decomposition of Audio Signals With Matching Pursuit". In: (2003).

[2] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data.* Oxford University Press, 2013.

[3] A Papandreou-Suppappola. *Applications in Time-Frequency Signal Processing (1st ed.)* CRC Press, 2003.

Appendix A MATLAB Functions

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `fftn` is the discrete fourier transform for n dimensions.
- `fftshift` reorders vector based on center of spectrum
- `ifftn` returns inverse discrete Fourier transform
- `abs` returns absolute value
- `ones(m, n)` creates a matrix of ones with m x n dimensions

Appendix B MATLAB Code

```
% Amelia Nathan
% AMATH 482, Assignment 2

figure(1)
[y, Fs] = audioread('GNR.m4a'); % Guns N's Roses Sweet Child O' Mind
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);

figure(2)
[y2, Fs2] = audioread('Floyd.m4a'); % Pink Floyd Comfortably Numb
tr_floyd = length(y2)/Fs2; % record time in seconds
plot((1:length(y2))/Fs2,y2);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Comfortably Numb');
p8 = audioplayer(y2,Fs2);
% playblocking(p8);
%% Sweet Child O' Mine, Guns N Roses, Guitar
L = tr_gnr; % How long the song is
s = y.';
n = length(s); % arbitrary value;
% chosen as power of 2 for efficient execution; number of data points
x2 = linspace(-L,L,n+1); % Create a vector with equally spaced sections
t = x2(1:n);
% Rescale frequency domain to fit our spatial domain
k = (1/L)*[0:(n/2 - 1) -n/2:-1]; % Changing this to hertz
ks = fftshift(k); % shift frequencies back to center of spectrum


% Sliding window across domain

a = 300;
tau = 0:0.05:14; % can make this smaller

for j = 1:length(tau)
   g = exp(-a*(t - tau(j)).^2); % Window function
   Sg = g.*s;
   Sgt = fft(Sg);
   Sgt_spec(:,j) = fftshift(abs(Sgt)); % We don't want to scale it
end

figure(3)
pcolor(tau,ks,Sgt_spec)
shading interp
set(gca,'ylim',[0, 1000],'Fontsize',16)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')
yline(220, 'w', '220: A')
yline(277, 'w', '277.183: C#')
yline(369, 'w', '369.99: F#')
yline(415, 'w', '415.305: G#')
yline(554, 'w', '554.36: C#')
yline(698, 'w', '698.456: F')
yline(739, 'w', '739.989: F#')
title('Guitar in Sweet Child O Mine')
```

```
%% Comfortably Numb, Pink Floyd Pt. 1
L = tr_floyd*0.5; % How long the song is; using two half song increments
s = y2.';
s = s(1:1317960); % raw signal
n = length(s);
x2 = linspace(-L,L,n+1); % Create a vector with equally spaced sections
t = x2(1:n);
% Rescale frequency domain to fit our spatial domain
k = (1/L)*[0:(n/2 - 1) -n/2:-1]; % Changing this to hertz
ks = fftshift(k); % shift frequencies back to center of spectrum


% Sliding window across domain

a = 300;
tau = 0:0.25:30;

for j = 1:length(tau)
   g = exp(-a*(t - tau(j)).^2); % Window function
   Sg = g.*s;
   Sgt = fft(Sg);
   Sgt_spec(:,j) = fftshift(abs(Sgt)); % We don't want to scale it
end

figure(4)
pcolor(tau,ks,Sgt_spec)
shading interp
set(gca,'ylim',[0, 1000],'Fontsize',16)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')
title('Floyd, Comfortably Numb, First Half')
yline(123, 'w', '123.47: B')
yline(220, 'w', '220: A')
yline(277, 'w', '277.183: C#')
yline(369, 'w', '369.99: F#')
yline(415, 'w', '415.305: G#')
yline(554, 'w', '554.36: C#')
yline(698, 'w', '698.456: F')
yline(739, 'w', '739.989: F#')
%% Comfortably Numb, Pink Floyd Pt. 2

L = tr_floyd*0.5; % How long the song is; using two half song increments
s = y2.';
s = s(1317961:2635920); % s is raw signal; using second half of song here
n = length(s); % arbitrary value;
% chosen as power of 2 for efficient execution; number of data points
x2 = linspace(-L,L,n+1); % Create a vector with 65 equally spaced sections
t = x2(1:n);
% Rescale frequency domain to fit our spatial domain
k = (1/L)*[0:(n/2 - 1) -n/2:-1]; % Changing this to hertz
ks = fftshift(k); % shift frequencies back to center of spectrum
```

```
% Sliding window across domain

a = 200;
tau = 0:0.25:30;

for j = 1:length(tau)
   g = exp(-a*(t - tau(j)).^2); % Window function
   Sg = g.*s;
   Sgt = fft(Sg);
   Sgt_spec(:,j) = fftshift(abs(Sgt)); % We don't want to scale it
end

figure(5)
pcolor(tau,ks,Sgt_spec)
shading interp
set(gca,'ylim',[0, 1000],'Fontsize',16)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')
title('Floyd, Comfortably Numb, Second Half')

xticklabels({'30', '35', '40', '45', '50', '55','60'})
yline(123, 'w', '123.47: B')
yline(220, 'w', '220: A')
yline(277, 'w', '277.183: C#')
yline(369, 'w', '369.99: F#')
yline(415, 'w', '415.305: G#')
yline(554, 'w', '554.36: C#')
yline(698, 'w', '698.456: F')
yline(739, 'w', '739.989: F#')

%% Part 2 - ISOLATING BASS - pt.1
% Run this section for both halves of the song

% Apply filter in frequency space
floyd_fft = fft(s);
shannon = ones(1, length(s)).*(abs(ks) < 200);
floyd_fft = floyd_fft.*fftshift(shannon);
floyd_fft = ifft(floyd_fft);


a = 250;
tau = 0:0.25:30;

for j = 1:length(tau)
   g = exp(-a*(t - tau(j)).^2); % Window function
   Sg = g.*floyd_fft;
   Sgt = fft(Sg);
   Sgt_spec(:,j) = fftshift(abs(Sgt)); % We don't want to scale it
end
```

```
figure(6)
pcolor(tau,ks,Sgt_spec)
shading interp
set(gca,'ylim',[0, 250],'Fontsize',16)
% Adjusted y limits to center and focus on bass
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')
yline(87, 'w', '87.307: F', 'LineWidth', 1)
yline(98, 'w', '97.999: G')
yline(110, 'w', '110.00: A')
yline(123, 'w', '123.47: B')
title('Comfortably Numb Bass, First Half')
%% Part 2 - ISOLATING BASS - pt.2

% Apply filter in frequency space
floyd_fft = fft(s);
shannon = ones(1, length(s)).*(abs(ks) < 200);
floyd_fft = floyd_fft.*fftshift(shannon);
floyd_fft = ifft(floyd_fft);


a = 250;
tau = 0:0.25:30;

for j = 1:length(tau)
   g = exp(-a*(t - tau(j)).^2); % Window function
   Sg = g.*floyd_fft;
   Sgt = fft(Sg);
   Sgt_spec(:,j) = fftshift(abs(Sgt)); % We don't want to scale it
end

figure(6)
pcolor(tau,ks,Sgt_spec)
shading interp
set(gca,'ylim',[0, 250],'Fontsize',16)
% Adjusted y limits to center and focus on bass
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')
xticklabels({'30', '35', '40', '45', '50', '55','60'})
yline(87, 'w', '87.307: F')
yline(98, 'w', '97.999: G')
yline(110, 'w', '110.00: A')
yline(123, 'w', '123.47: B')
title('Comfortably Numb Bass, Second Half')
%% ISOLATING GUITAR - pt.1

% Apply filter in frequency space
floyd_fft = fft(s);
shannon = ones(1, length(s)).*(abs(ks-350) < 150);
floyd_fft = floyd_fft.*fftshift(shannon);
floyd_fft = ifft(floyd_fft);
```

```
a = 200;
tau = 0:0.2:30;

for j = 1:length(tau)
   g = exp(-a*(t - tau(j)).^2); % Window function
   Sg = g.*floyd_fft;
   Sgt = fft(Sg);
   Sgt_spec(:,j) = fftshift(abs(Sgt)); % We don't want to scale it
end

figure(7)
pcolor(tau,ks,Sgt_spec)
shading interp
set(gca,'ylim',[200, 500],'Fontsize',16)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')
yline(246, 'w', '246.94: B')
yline(277, 'w', '277.18: C#')
yline(369, 'w', '369.99: F#')
yline(415, 'w', '415.305: G#')
title('Comfortably Numb Guitar, First Half')
%% ISOLATING GUITAR - pt.2

% Apply filter in frequency space
floyd_fft = fft(s);
shannon = ones(1, length(s)).*(abs(ks-350) < 150);
floyd_fft = floyd_fft.*fftshift(shannon);
floyd_fft = ifft(floyd_fft);


a = 200;
tau = 0:0.2:30;

for j = 1:length(tau)
   g = exp(-a*(t - tau(j)).^2); % Window function
   Sg = g.*floyd_fft;
   Sgt = fft(Sg);
   Sgt_spec(:,j) = fftshift(abs(Sgt)); % We don't want to scale it
end

figure(7)
pcolor(tau,ks,Sgt_spec)
shading interp
set(gca,'ylim',[200, 500],'Fontsize',16)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')
xticklabels({'30', '35', '40', '45', '50', '55','60'})
yline(220, 'w', '220: A')
yline(277, 'w', '277.183: C#')
yline(369, 'w', '369.99: F#')
yline(415, 'w', '415.305: G#')
title('Comfortably Numb Guitar, Second Half')
```