Reproducing Music Scores with Gabor Transform

Johnny Van

February 3rd, 2020

Abstract

Using the Gabor Transform, we analyzed audio clips of two famous songs and were able to derive the frequencies of the notes being played by particular instruments. Then, since we know the exact frequencies of the notes, we were able to recreate the music score for that instrument. Also, in this project, we were able to isolate certain instruments from the audio clips, allowing us to play filtered clips that contain only that instrument.

1 Introduction and Overview

If we were given a simple audio file of a clip from a song like Gun's and Roses, could we somehow figure out exactly what notes are being played at a specific time? If we were music enthusiasts, it could be possible by ear. However, this project covers techniques that can be used to recreate music scores of music clips. We are given the audio files of two famous songs, "Sweet Child O' Mine" by Guns and Roses(GNR), and "Comfortably Numb" by Pink Floyd. Using the Gabor Transform we can analyze both frequency and time information from the audio clips and create its spectrogram in order to derive exactly what notes are being played. Also, we are able to isolate certain instruments from the clips and even be able to play clips that contain only the instrument.

2 Theoretical Background

If we are given an audio signal and plot it as amplitude of the signal vs. time, is there a way to break it down into its frequencies? Well if we were given a function of signal vs. time, we would break it down into sines and cosines of different frequencies. First let's mention Euler's formula,

$$e^{i,\theta} = cos(\theta) + isin(\theta) \tag{1}$$

Using Euler's formula we can convert formulas that utilize sines and cosines into formulas with complex exponentials back and forth.
Then, by the textbook, [1], we are introduced to the Fourier Transform. Suppose we are given a function f(x), we can define the Fourier Transform of f(x) as $\hat{f}(k)$ and its formula is defined by Equation (2).

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

The Fourier Transform takes a function of space/time and converts it into a function of frequencies. Then, if instead you are given $\hat{f}(k)$ and want to find f(x), then you can utilize the inverse Fourier transform to do so, and effectively changing the function's domain back to the space/time instead of frequency. The Inverse Fourier Transform is given by Equation (3).

$$f(x) = \frac{1}{\sqrt{2\pi}} \int \hat{f}(k) e^{ikx} dx \tag{3}$$

From Euler's formula, since we know that e^{ikx} is similar to sin(kx) and cos(kx), the value of k determines the frequences of these sine and cosine waves. So, we can see that the Fourier Transform takes a function of space(or time), x, and converts it to a function of k, or frequency.

2.0.1 Limitations of the Fourier Transform

There are some caveats to the Fourier transform. The raw data we receive tells us everything about space and time, but nothing about frequency information. When we perform a Fourier transform on the data, we receive information about whether or not a specific frequency exists, but it does not tell when this frequency occurs. So whenever we are playing with the data, we are always in either the space/time domain or the frequency domain, so we back and forth in order to uncover information about both. Using the Fourier Transform works great with periodic signals, but doesn't work well with non-stationary signals. A technique that gives information about both time and frequency information is the Gabor transform, which is better suited for this job than the standard Fourier Transform.

2.0.2 Gabor Transform

First let's consider filter function g(t). Since the Gaussian filter is the standard filter, let's say g(t) is the Gaussian filter for this example. Then we shift this filter by τ and multiply it by a function f(t), and this represents a filtered function $f(t)g(t-\tau)$, with filter centered at τ. The Gabor Transform then is given by Equation 4.

$$\tilde{f}_g(\tau, k) = \int_{-\infty}^{\infty} f(t)g(t-\tau)e^{-ikt}dt \tag{4}$$

So for a set τ, the Gabor Transform, $\tilde{f}_g(\tau, k)$, gives you information about frequency near time τ. Of course, depending on the filter you choose, you will get different results for the Gabor Transform. There are two common assumptions about the filter g(t).

1. The function g(t) is real and symmetric
2. $||g||_2 = (\int |g(t)^2|dt)^{\frac{1}{2}} = 1$, that is the L_2-norm of g is set to unity.

Then, of course we have the inverse Gabor Transform, given by Equation 3.

$$f(t) = \frac{1}{2\pi||g||_2} \int_{-\infty}^{\infty} \tilde{f}_g(\tau, k)g(t-\tau)e^{ikt}dkd\tau \tag{5}$$

We have another parameter consider, the width of the filter used in Gabor Transform. If we set the window too large, then we get lots of frequency information but too little time information. On the other hand, if we set the filter window too small, then we get time information but little frequency information. So when using the Gabor Transform, you have to find a sweet spot, or a window width that is not too large or small.

2.0.3 Discrete Gabor Transform

Similarly to the Fourier Transform, if we want to use the Gabor Transform on data, we must use a discrete version. So with the Gabor Transform, the filter is centered at τ. To use the Gabor Transform on data, we would essentially drag the filter across the entire signal by shifting τ by a certain amount and taking the Gabor Transform at that point, and keep repeating this process until τ has reached its limit. For example, τ could start at 0 and move in steps of 0.1 until it reaches 14. Note, we have to consider only a discrete set of frequencies.

$$k = m\omega_0$$

$$\tau = nt_0$$

where m and n are integers and ω_0 and t_0 are positive constants representing the frequency resolutions. Thus the discrete Gabor Transform has the form,

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

2.0.4 Spectrogram

The Gabor Transform is basically sliding a filter over the entire signal and taking the Fourier Transform of the data under this filter as it slides along the domain. We can't show the video of the sliding filter in a report, it is easier to convey our results in a spectrogram. Basically we stack all the Fourier transforms back to back to make a plot with the horizontal direction being τ, and vertical direction being the frequency range. Using a spectrogram allows us to create a still image that shows the changing of the Fourier Transform as the filter slides over the domain. The brightness of the spots on the spectrogram would then indicate the magnitude of that frequency at a certain time.

3 Algorithm Implementation and Development

3.1 Data Preparation

It is important to understand that the audio signals of the music clips are given to us in vectors, a 1 by some very large number vector. The vector's length, or number of data points depend on the length of the audio clip. Before we implement our algorithmns on the data. it is very important to correctly set up the time and frequency domains of our data. The first variable we need to define is L, which is the length of the audio clip in seconds, and we also define the number of fourier modes, n, to equal the number of data points in the vector of the audio clip. Using these two variables, I created the frequency domain based on the number of fourier modes and created time domain using the length of the audio clip. You can see this in Appendix C. Also, if the clip is too long, I used an audio trimmer to load in segments of the clip into Matlab.

Note: When creating the frequency domain, we will want to scale it by 1/L in order to have the scale used for frequencies in Hertz. Scaling frequency domain by $\frac{2\pi}{L}$ will give angular frequency.

3.2 Algorithm 1: Creating Spectrograms of each Clip

First, on the GNR clip, we took the Gabor transform of the audio vector first and created a spectrogram based off of it.

1. Prepare the audio file by converting it into vector form.
2. Perform the Gabor Transform on the audio vector.
    a) Multiply the audio vector by a Gaussian filter, centered at τ, with a width of a.
    b) Perform the Fourier Transform on this filtered vector.
    c) Add the transformed filtered vector to the spectrogram array.
    d) Perform the above 3 steps until you went through all steps of τ.
3. Using the spectrogram array, create the spectrogram using a heatmap configuration.
4. Using the frequency information from the spectrogram, label the notes being played.
5. If the spectrogram does not look clear, change the width of the filter, a, or the steps of τ to a larger or shorter size.

3.3 Overtones/Harmonics

If we look at figure 2 for example, we can almost see an exact copy of the notes being played above the actual notes of the bass guitar. This phenomenon is referred to as the overtone or harmonics of the bass. Overtone is usually predictable because the overtone of a note occurs at an integer multiple of the note's frequency. For example, if the frequency of the note is 90 hertz, then the overtones would usually occur at the hertz of 180, 270, and so on. The overtone of the bass notes, however, does not appear at integer multiples of the original notes because the bass guitar is a special case. The B2 note has a frequency of around 120 hertz, and we can see its overtone at around 190 which indicates that it its overtone occurs at about 1.5 times the frequency of the original note.

Note: The Gaussian function could be described as the default filter for the Fourier Transform and Gabor Transform, but it is not well-suited for the upcoming task. In this case, we will begin to use the Shannon filter, which looks like a rectangular box that when multiplied to data, only keeps the data that lies within this box.

3.4 Algorithm 2: Isolating the bass from Comfortably Numb

From part 1, we created the spectrograms for both clips. In the Comfortably Numb spectrogram, since we assume that the bass guitar has lower frequencies than the guitar, we can deduce that the range of frequencies for the bass guitar is around [50, 200] hertz. Using this information, we simply filter this range of frequencies out from the entire audio vector in order to isolate it, then run the filtered audio vector through the Gabor Transform to create the spectrogram.

1. Prepare the audio file by converting it into vector form.
2. Peform the 1-D Fourier Transform on the audio vector to enter the frequency domain.
3. Create a Shannon filter than contains the range of frequencies you want.
4. Multiply the transformed data by a shifted Shannon filter.
5. Perform a inverse 1-D fourier transform on the filtered data, reverting the data back to the time domain. This filtered audio vector should only contain signal information from the bass guitar.
6. Run this audio vector through Algorithm 1, starting from Step 2.

3.5 Algorithm 3: Filtering out overtones

Filtering out the overtones is very similar to isolating the bass in Algorithmn 2. Instead of creating Shannon filters that contain a large range of frequencies such as [200,500], you create multiple shannon filters that contain a very small range. For example, lets say that a note is played by an instrument and has 120 hertz. So we know there are overtones with frequencies of 240,360, and so on. So we would create Shannon filters around each of these overtones with a width of about 5 hertz and filter them out of the data in the frequency domain. Then take this filtered data and take the 1-D inverse Fourier transform to convert the data back into the time domain. This data now just contains all the original audio data except all of the overtones that were filtered out.

4 Computational Results

4.1 Part 1: Spectrogram of GNR and Comfortably Numb

Using Algorithmn 1, I created the spectrograms of both audio clips and adjusted my filters enough to produce excellent figures. In figure 1, we can see the spectrogram of GNR and it's notes are labelled on the y-axis so we know exactly what notes are being played at what time.

In Comfortably Numb, both the bass guitar and guitar are being played. Since the bass has a lower frequency, we deduced that the bass likely has notes around the range of [50,200] and the guitar's notes are in the range of [200,800]. So you can see in figure 2, I labelled all the notes played by the bass in "Comfortably Numb", and they have rather low frequency. You can also see a lot of overtone,, which looks like a faded out version of the actual notes played. The spectrogram for the first 30 seconds of the clip are in figure 2, the rest of the clip is shown in figure 4.

4.2 Part 2: Isolated Bass of Comfortably Numb

Using Algorithm 2 and assuming that the bass guitar notes have a frequency range of [0,200], I was able to isolate the audio signal of the bass from the entire audio vector. Then I created its spectrogram in figure 3. Notice that if you compare figure 3 to figure 2, that we managed to cut out all frequencies above 200 hertz in

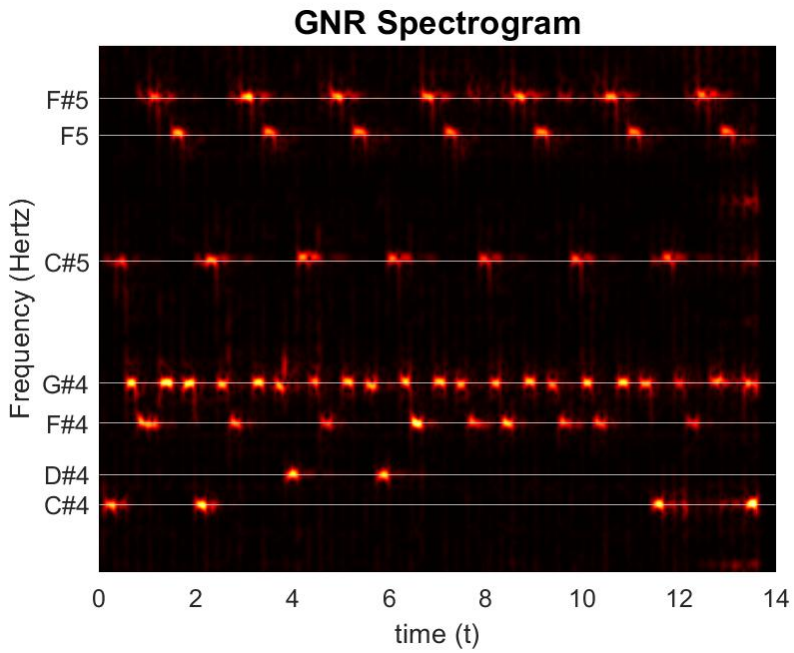

Figure 1: Spectrogram of entire Guns and Roses clip and notes played by the guitar are labelled.

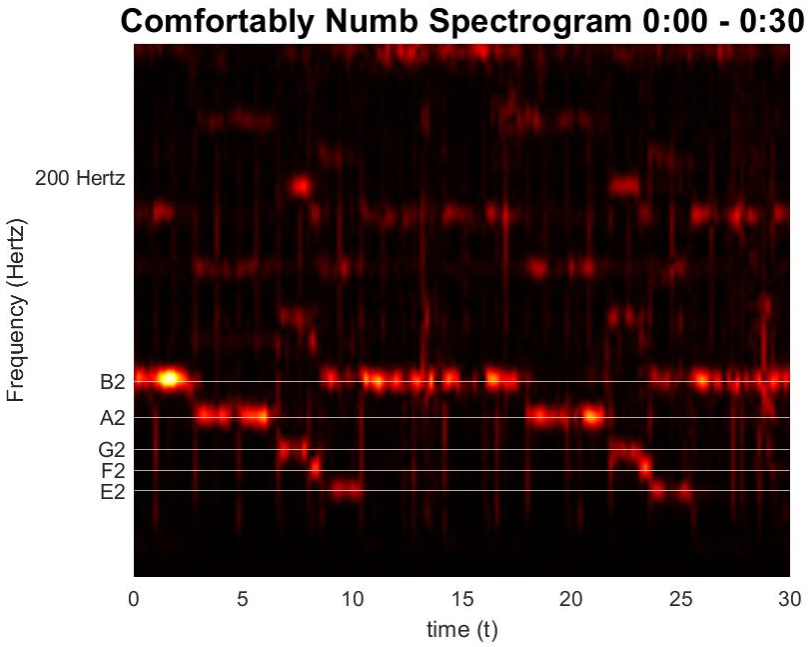

Figure 2: Spectrogram of Comfortably Numb and notes played by the bass guitar are labelled. Covers 0:00 to 0:30 of the audio clip.

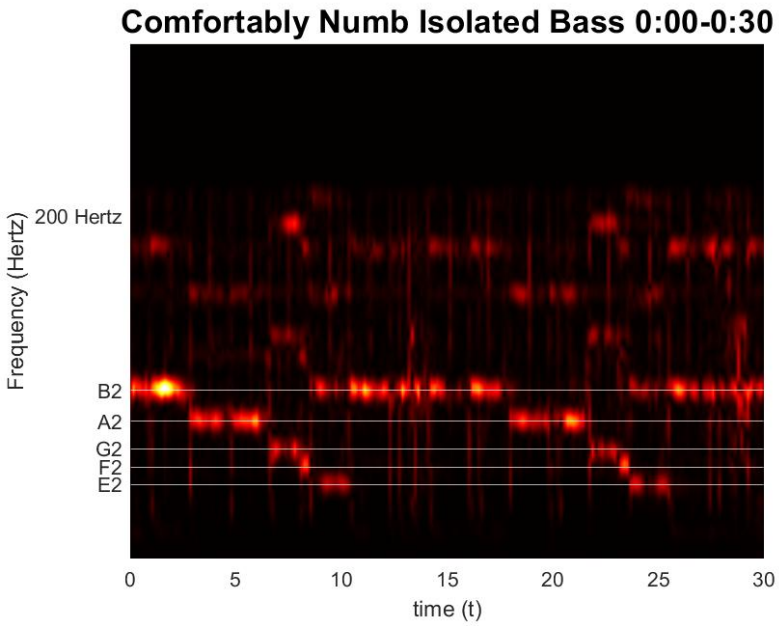

Figure 3: Spectrogram of "Comfortably Numb" with bass guitar isolated. Covers 0:00 to 0:30 of the audio clip. The notes played by the bass are labelled.

figure 3. This means that we were successful in completely isolating the bass from the original audio vector. Also, since we isolated the bass guitar into its own audio vector, we are able to use MATLAB in order to put this vector through a music player and hear only the bass guitar playing.

4.3 Reconstruction of the Guitar Solo in Comfortably Numb

Using Algorithmn 2 and 3, we tried to isolate the guitar in Comfortably Numb like we did with the bass. The main obstacle in doing this, is the presence of the bass guitar overtones as you can see in figure 5 in Appendix A. The spectrogram becomes very messy in the range of [200,800] because it contains the notes played by guitar but also the overtones of the bass guitar as well. Using Algorithm 3, we filtered out as much of the bass guitar overtones as possible, then ran the resulting audio vector through Algorithm 2 in order to isolate the range of [200,800] and plot its spectrogram which you can see as figure 6 in Appendix A. I could only identify a few notes such as a a D5 note with frequency of 590 hertz, C5 note (660 hertz) and D4 note (400 hertz). These are probably not accurate due to how much overtone is present which makes it very difficult to reproduce the music score of the guitar solo. But, we did make an effort to do.

5 Summary and Conclusions

Given the audio clips of music, we were able to use the Gabor Transform on these audio clips to accurately uncover both time and frequency information. This allowed us to deduce the exact notes being played by certain instruments based on their spectrograms. Furthermore, if we use a Shannon filter on the signal to isolate a certain range of frequencies, then use the Gabor transform on it, we can essentially isolate certain instruments from audio clips and even be able to play clips of the same song, but just with the instrument we isolated! Very incredible work as this could apply to many real-world situations.

References

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

Appendix A Supplementary Figures

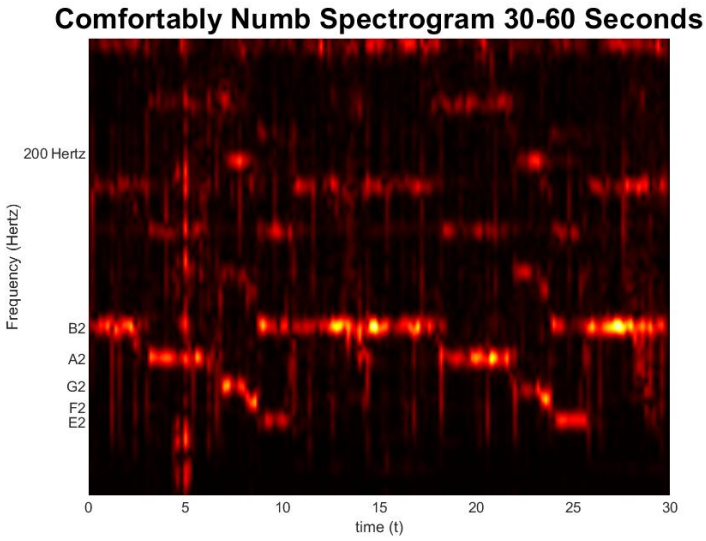

Figure 4: Spectrogram of Comfortably Numb that covers 0:30 to 1:00 seconds of the audio clip. The notes played by the bass are labelled.

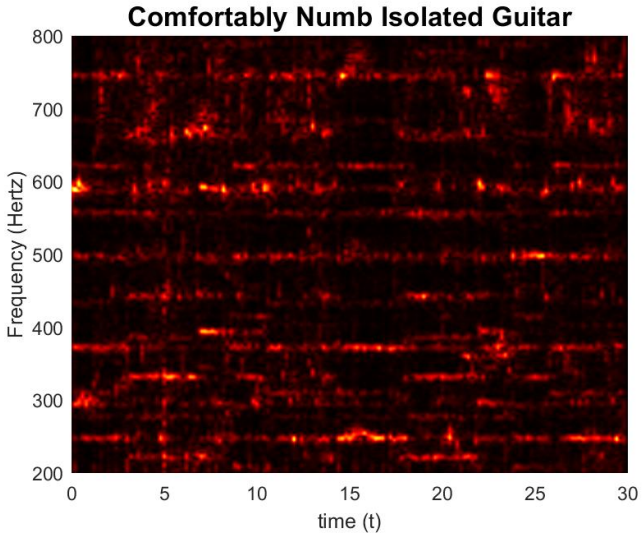

Figure 5: Spectrogram of Guitar Solo in Comfortably Numb with overtones present. Covers 0:30 - 0:60 of the Comfortably Numb clip

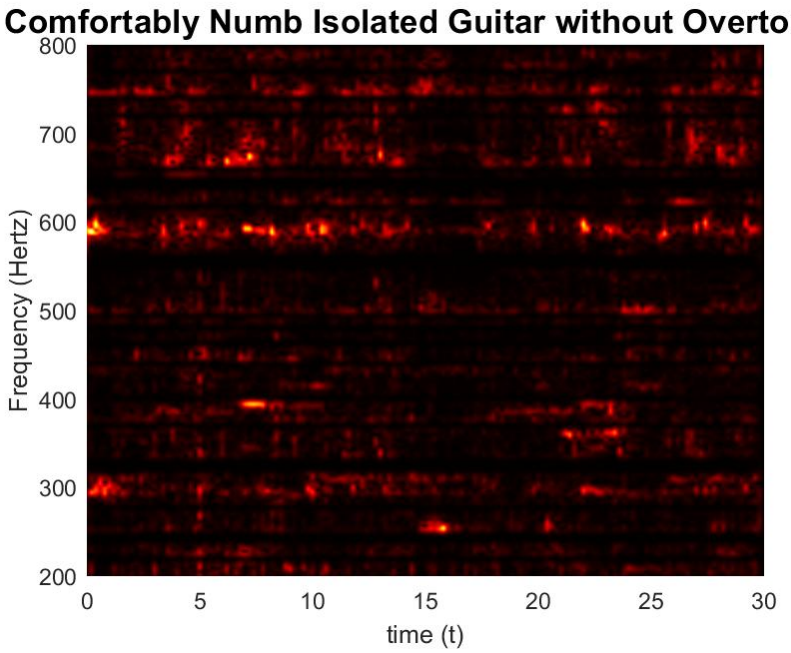

Figure 6: Spectrogram of Guitar in Comfortably Numb with overtones filtered out. Covers 0:30 to 1:00 of audio clip.

Appendix B MATLAB Functions

Add your important MATLAB functions here with a brief implementation explanation. This is how to make an **unordered** list:

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `Y = fft(X)` returns the 1-D Fourier Transform of a vector X.
- `Y = fftshift(X)` swaps the right and left halves of a vector X.
- `Y = ifft(X)` returns the inverse 1-D Fourier Transform of a vector X.
- `[Max, Index] = max(x(:))` returns the max of an array of values and gives its index.
- `pcolor(X,Y,C)` returns a pseudocolor plot where vectors X and Y define the x and y coordinates of the grid respectively and matrix C determines the color at each of those coordinates.

Appendix C MATLAB Code

```
%%
clear all; close all;

figure(1)
[y, Fs] = audioread('GNR.m4a'); %convert clip into vector
trgnr = length(y)/Fs; % length of clip

S = y.'; % transpose of audio vector
n = length(S); % length of audio vector
L = trgnr; % length of time domain
t2 = linspace(0,L,n+1); t = t2(1:n); % Create time domain
k = (1/L)*[0:n/2-1 -n/2:-1]; ks = fftshift(k); % Create frequency domain
                                               % Scale by 1/L for Hertz


%% Part 1(GNR Spectrogram)

a =300; %width of Gaussian filter
tau = 0:0.05:14; % the chosen centers for filter

for j = 1:length(tau) % Gabor Transform
    g = exp(-a*(t - tau(j)).^2); % Gaussian filter function
    Sg = g.*S; % Applying filter to audio vector
    Sgt = fft(Sg); % 1-D Fourier Transform of audio vector
    Sgtspec(:,j) = fftshift(abs(Sgt)); % Creates Spectrogram matrix
                                       % (tau x transformed data)

end

pcolor(tau,ks,Sgtspec) % Creates Spectrogram
shading interp
set(gca,'Fontsize',12)
colormap(hot) % Sets spectrogram type to a heatmap look
yline(277.17, 'w');yline(311.12, 'w');yline(369.98, 'w');
yline(415.29, 'w');yline(554.36, 'w');yline(698.45, 'w');
yline(739.98, 'w'); % Creates horizontal lines at guitar notes
xlabel('time (t)'), ylabel('Frequency (Hertz)')
yticks([277.17, 311.12, 369.98, 415.29, 554.36,698.45,739.98]);
yticklabels({'C#4', 'D#4', 'F#4', 'G#4','C#5','F5','F#5'}); %label guitar notes on y-axis
ylim([200 800])
title(['GNR Spectrogram'],'Fontsize',16);
saveas(gcf,'GNR Report.jpg')
%% Part 1(Floyd Spectrogram)

a =150;
tau = 0:0.2:30;

for j = 1:length(tau)
    g = exp(-a*(t - tau(j)).^2); % Gaussian filter function
    Sg = g.*S;
    Sgt = fft(Sg);
    Sgtspec(:,j) = fftshift(abs(Sgt));
```

```
end

pcolor(tau,ks,Sgtspec)
shading interp
set(gca,'ylim',[50, 250],'Fontsize',10)
colormap(hot)
yline(82.41,'w'); yline(90.00,'w'); yline(98.00,'w'); yline(110.00,'w');
yline(123.47,'w');
xlabel('time (t)'), ylabel('Frequency (Hertz)')
yticks([82.41,90.00,98.00, 110,123.47, 200, 300]);
yticklabels({'E2', 'F2','G2','A2', 'B2','200 Hertz', '300 Hertz'});
title(['Comfortably Numb Spectrogram 0-30 Seconds'],'Fontsize',16);
saveas(gcf,'Comfortably Numb Spectrogram 0-30 Report.jpg')


%% Part 2 (Isolating the Bass Guitar in Comfortably Numb)

rect=@(x,a) ones(1,length(S)).*(abs(x)<a/2) % Creates a rectangular
                                             % function

shannon_filter = rect(ks,420); % Creates shannon filter centered at 0, width 420.

S_gt = fft(S); % 1-D Fourier Transform on entire audio vector
S_filter = S_gt.*fftshift(shannon_filter); % Filter out everything not
                                           % contained in filter.

S_inv = ifft(S_filter); % Take the inverse of the filtered data to
                        % convert back into time domain. Put through
                        % Gabor Transform

a =200; % width of Gaussian filter
tau = 0:0.15:30; % Centers of Gaussian filter

for j = 1:length(tau)
    g = exp(-a*(t - tau(j)).^2); % Gaussian Filter function
    Sg = g.*S_inv;
    Sgt = fft(Sg);
    Sgtspec(:,j) = fftshift(abs(Sgt));

end

pcolor(tau,ks,Sgtspec)
shading interp
set(gca,'ylim',[50, 275],'Fontsize',10)
colormap(hot)
yline(82.41,'w'); yline(90.00,'w'); yline(98.00,'w'); yline(110.00,'w');
yline(123.47,'w');
xlabel('time (t)'), ylabel('Frequency (Hertz)')
yticks([82.41,90.00,98.00, 110,123.47, 200, 300]);
yticklabels({'E2', 'F2','G2','A2', 'B2','200 Hertz', '300 Hertz'});
title(['Comfortably Numb Isolated Bass 0-30 Seconds'],'Fontsize',16);
saveas(gcf,'Comfortably Numb Isolated Bass Spectrogram.jpg')
```

```
%% Part 3 (Guitar Solo Isolation)

rect=@(x,a) ones(1,length(ks)).*(abs(x)< a/2) % a is the width of the pulse

shannon_filter = rect(ks-500,600); Shannon filter center at 500, width of 600.

S_gt = fft(S); % 1-D Fourier Transform on data

notes = [80.00,92.50,110,123.47]; % Notes of Bass Guitar from Part 1
for l = 1:10
    for p = 1:4
        filter = rect(ks-l*notes(p),10); % Creates filter around bass notes
        S_f = S_gt.*fftshift(filter);
        S_gt = S_gt-S_f; % Subtract filtered data from overall transformed data.
    end
end

S_isolated = S_gt.*fftshift(shannon_filter); % Isolate frequency range of [200,800]
S_inv = ifft(S_isolated); % 1-D Inverse Fourier Transform on filtered data to
                          % revert back to time domain. This contains no overtones
                          % and should only contain guitar notes.

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

for j = 1:length(tau)
    g = exp(-a*(t - tau(j)).^2); % Window function
    Sg = g.*S_inv;
    Sgt = fft(Sg);
    [Max, Index] = max(abs(Sgt)); % Finds the index of frequency with max magnitude, Optional Step
    Guitar_notes(1,j) = abs(k(Index)); % Finds the frequency with max magnitude. Optional Step
    Sgtspec(:,j) = fftshift(abs(Sgt));

end

plot(tau,Guitar_notes,'o','MarkerFaceColor', 'b') % Optional, plots the max
                                                  % max frequency at each time step.
set(gca,'ylim',[0, 1000],'Fontsize',8)
title("Score for Guitar Solo")
xlabel('time (t)'), ylabel('Frequency (Hertz)')

pcolor(tau,ks,Sgtspec)
shading interp
set(gca,'ylim',[200, 800],'Fontsize',11)
colormap(hot)
xlabel('time (t)'), ylabel('Frequency (Hertz)')
title(['Comfortably Numb Isolated Guitar'],'Fontsize',16);
saveas(gcf,'C.N. Isolated Guitar.jpg')
```