AMATH 482 Homework 2

Haley Riggs

January 28, 2020

Abstract

Numerous filtering techniques exist and are useful for different applications. In this paper, we utilized the Gabor transform, paired with a Gaussian filter and a Shannon filter to recreate music scores for the guitar and bass parts of the songs "Sweet Child O' Mine" and "Comfortably Numb."

1 Introduction and Overview

In this assignment, we used clips of the songs "Sweet Child O' Mine" by Guns 'n Roses and "Comfortably Numb" by Pink Floyd to reproduce associated music scores. Through filtering techniques, we assembled the guitar part for the Guns 'n Roses song, the bass part for the Pink Floyd song, and attempted to reconstruct the music score for the guitar solo in "Comfortably Numb." Overtones were filtered out to obtain a cleaner score.

2 Theoretical Background

As discussed in the last assignment, the Fourier transform is a useful tool in analyzing the frequency of a signal. However, due to shift invariance, the Fourier transform loses the data's time information. Despite gaining insight into the different frequencies occurring within the signal, we can't really tell when these frequencies take place or how they change in time. We will now consider how we can obtain both time and frequency information about a signal.

We can first try to divide the entire time domain into parts, taking the Fourier transform of each section. While this enables us to gain a bit more information about the frequencies that occur within each block of time, we remain unsure about what happens across the span of multiple regions. We have also introduced sharp transitions between blocks of time and thus have basically assumed that the signal outside of each window, the signal is zero. To counteract these problematic properties, we can slide a time filter across the entire domain, examining every possible location for it. This is the idea behind the Gabor transform, which is also referred to as the short-time Fourier transform (STFT).

If we let $g(t)$ denote a filter function, $f(t)$ be a function, and τ describe the filter's center then the filtered function can be represented by $f(t)g(t-\tau)$. The Gabor transform is then defined by:

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

Considering a particular value of τ, $\tilde{f}_g(\tau, k)$ describes the frequency components of the signal close to time τ. These results depend on the filter function $g(t)$, which is often assumed to have the following properties:

1. $g(t)$ is real and symmetric
2. $||g||_2 := \int_{-\infty}^{\infty} |g(t)|^2\, dt)^{\frac{1}{2}} = 1$ (This means that the L_2-norm of g, which represents g's total energy, is set to unity. This is a condition that ensures that the energy is neutral.)

These properties make he inverse of the Gabor transform, which is denoted in the following equation, simpler to compute.

$$\frac{1}{2\pi||g||_2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\tilde{f}_g(\tau,k)g(t-\tau)\mathrm{e}^{\mathrm{i}kt}\,dk\,d\tau \tag{2}$$

g can be set to anything, as long as it satisfies the listed properties. However, since Gabor used a Gaussian when he developed this method, our class has decided that our default will be the Gaussian as well. (Technically, the STFT refers to this process with a general $g(t)$ and the Gabor transform specifically uses a Gaussian as $g(t)$.)

The width of the window, which is denoted as a if we allow our Gaussian to be generally defined as $g(t) = \mathrm{e}^{-a(t-\tau)^2}$, must also be considered when using the Gabor transform. Its value changes depending upon the context of the problem, but the following must be considered:

- If we choose a very large window, we essentially obtain the Fourier transform over the entire domain, which gives us all of the frequency information but no time information.
- If we choose a very small window, we obtain all of the time information but no information about the specific frequencies.

Therefore, a balance must be struck between the two. This brings us back to the Heisenberg uncertainty principle, discussed in the last homework assignment, which tells us that when more information about the time domain is obtained, we will get less information about the frequencies. Alternatively, if more information regarding the frequencies is obtained, we know less about the time domain. We must compromise between the two. In terms of implementation, it is important to note that there is an inverse relationship between the value of a and the width of the window.

Similar to what was discussed for the Fourier transform, it is necessary to have a discrete definition for the Gabor transform in order to use it on data. For practical purposes, we cannot define τ to be some arbitrary value; it must be a discrete set of values. We also need a discrete set of frequencies which, if we let m and n be integers and ω_0 and t_0 represent the frequency resolutions and are positive, constant values, we define as:

$$k = m\omega_0 \qquad \tau = nt_0 \tag{3}$$

The discrete Gabor transform is then:

$$\tilde{f}_g(m,n) = \int_{-\infty}^{\infty} f(t)g(t-nt_0)\mathrm{e}^{2\pi\mathrm{i}m\omega_0 t}\,dt \tag{4}$$

Also note that in order to reproduce a signal, it is necessary to have a large enough window, or small enough time-step t_0 to have a little bit of overlap. Additionally, for quality frequency resolution, we require a small enough ω_0.

In addition to the Gaussian, another filter of use is the Shannon filter. It is a step function, equalling either 0 or 1. While the Gaussian is a smooth function, the Shannon filter has a sharp cutoff in frequency space, setting all frequencies above some threshold value to be zero.

3 Algorithm Implementation and Development

In this assignment, audio files of the songs "Sweet Child O' Mine" from Guns 'n Roses and "Comfortably Numb" by Pink Floyd were imported into MATLAB and then processed to reconstruct different instruments' parts of the song. To start, data was uploaded into MATLAB using the `audioread()` function, which processes an audio file and returns the associated sample data and a sample rate. The length of this vector was then divided by the sample rate to discover the length of the record in seconds. To better understand the data, a plot of the frequencies was created and the songs were played using the functions `audioplayer()` and `playblocking()`.

Next, for each song, variables were defined to set up the problem. The number of Fourier modes was set to be the length of the sample data vector. Time was discretized into a domain extending from 0 to the length of the recording, split into the number of Fourier modes plus one pieces. Even though the data is not periodic, it was treated as if it was, enabling the use of the Fourier transform later on. This meant that, if we let n be the number of Fourier modes, only the first n pieces of the time domain were utilized. Frequency components were also defined. However, instead of scaling by 2π divided by the length of the spatial domain as is usually done for the FFT, the 2π was left out to stay in Hertz. The frequency components were instead scaled by one divided by the length of the spatial domain. They were also shifted using `fftshift()`.

The first goal of this assignment was to reproduce the guitar's music score for "Sweet Child O' Mine" and the bass's music score for "Comfortably Numb." In order to do this, it was necessary to define a Gabor transform. For the Guns 'n Roses song, the different window centers, τ, were defined to be equally spaced increments between 0 and the length of the spatial domain. The window size was originally set to be 2 though, through visual observation and consideration of the necessary balance of time and frequency information, was adjusted to 1000, then 2000, and finally 1500. 1500 seemed to make it fairly easy to observe the frequencies of the notes being played while retaining time information.

Then, a for loop going from one to the length of the τ vector was defined. Inside of it was a Gaussian window function set to be $\mathrm{e}^{(t-\tau(j))^2}$ where t represented the time domain vector and $\tau(j)$ is the jth value in the τ vector. The Gaussian function was then multiplied by the sample data vector and put through the FFT. Its `fftshift`-ed absolute value was then stored in the jth column of a matrix. At the end of the for loop's cycle, this matrix contained the shifted, filtered data in frequency space at every window of the Gabor transform.

After the matrix of Fourier transforms was completed, its values were plotted in a spectrogram. This spectrogram essentially contained a stack of Fourier transforms, side by side, with the frequency information on the y-axis and the values of τ being the x-axis. It is a visualization of the filtered frequencies across time.

At first, I attempted to plot all of the data at once. I chose to adjust the y-axis limits to reflect where the base frequencies of the notes were. I noted that since this audio clip is primarily the guitar part, anything located above a strong frequency in the spectrogram was either noise or an overtone and for our purposes, should be ignored. This resulted in a very slow program.

In an attempt to speed the program up, I decided to constrict the range of frequency values instead of plotting everything and adjusting the y-axis limit. I discovered what the possible frequencies in Hertz a guitar could play were and used those values, roughly 75 to 1250 Hertz, as my limits. Knowing that the shifted frequency components increased in value, I found the index of the first value that was greater than 75 and then the index of the first value that was greater than 1250. Afterwards, I restricted the frequency components to be only those between those indices and restricted the rows in the matrix of spectrogram frequency data to be between those indices as well. Upon further examination, the more appropriate bounds (eliminating most overtones) were 75 and 800 Hertz. I changed my code respectively.

The next step was to examine the guitar sheet music for the Guns 'n Roses song and find the notes' conversions to Hertz. After I had done this, I added labels to the y-axis showing what the notes were and what their associated Hertz frequencies were.

A similar process was completed for the Pink Floyd song, considering the bass guitar part. Picking the window size to be 100 instead of 1500 seemed to provide a clearer spectrogram image. Additionally, since the clip is longer, I decided to take step sizes of one half from zero to the length of the spatial domain for the values for τ. Since the possible frequencies a bass guitar can play are between about 50 and 1050 Hertz, I found the indices of the first values in the shifted frequency components that corresponded to these. I then restricted the displayed frequency component values and the rows of the frequency data to reflect this. Through visual analysis, I realized that a more appropriate interval was from 50 to 150 Hertz so I changed the restrictions accordingly. I then examined the sheet music for the bass guitar part and the Hertz frequencies that corresponded to them and labeled the spectrogram accordingly.

Our next goal was to find the bass guitar part in the Pink Floyd song through the use of a filter in frequency space. As before, τ was set to range from zero to the length of the spatial domain in step sizes of one half. The width of the window was set to be 100.

The next goal was to find the bass part of the Pink Floyd song using a filter in frequency space. The window centers, a τ vector, were set to be values from zero to the length of the spatial domain in increments of one half. The width of the window was set to be 100.

At this point, a Shannon filter was used to keep the data for frequencies above 150 Hertz since all of the bass guitar notes were found to be within the range of 50 to 150 Hertz. The sample data vector was put into frequency space using the function `fft()`. Then, the values of this vector that has indices corresponding to

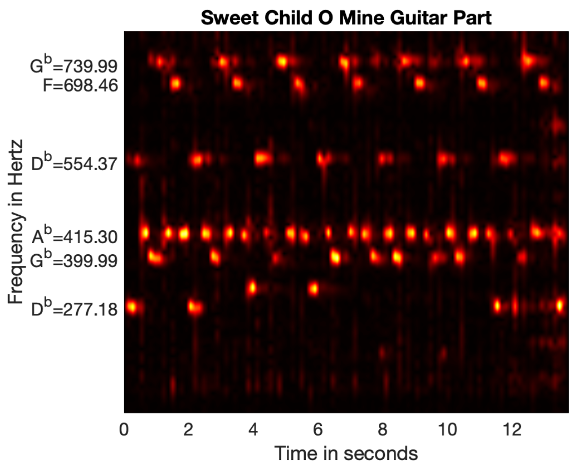

Figure 1: Spectrogram of the guitar's music score for the song "Sweet Child O' Mine" by Guns 'n Roses

Algorithm 1: Reproduce Music Score Through Restricting Domain

```
Extract data from song file
Define vector tau, window centers, and a, window size
for j = 1 : length(tau) do
    Define Gaussian window function
    Multiply filter by sample data array, element-wise
    Perform fft on array
    fftshift the absolute value of the array
    Insert array into column j of a matrix
end for
Find appropriate indices of shifted frequency components
Restrict values in frequency components
Restrict rows in matrix of filtered frequencies
Create spectrogram of results with notes labeled
```

the shifted frequency components greater than 150 Hertz were set to 0. The resulting vector was put back into the time domain using `ifft()`.

Afterwards, a Gabor transform was performed on the data. As before, a for loop running from one to the length of the τ vector was created. Inside the loop, a Gaussian window function with the same structure as before was created, shifting its center at each iteration of the loop to the next value of τ. The filter was multiplied element-wise by the data vector and put into the frequency domain. Its absolute value was calculated and shifted using `fftshift` and inserted into a column or a matrix, corresponding to the current iteration of the loop. The result was plotted in a spectrogram with the notes labeled.

Our last goal was to reproduce the guitar solo for "Comfortably Numb." The guitar part was not nearly as pronounced as the bass guitar part so this posed a huge challenge. My first thought was to impose a Shannon filter on the sample frequency data to find only the bass part, similar to what was done for the previous goal. The data would then be converted back to the time domain for the Gabor transform. Then, I would create a for loop, similar to what was described before. However, this time, after using a Gaussian window function to filter the data and transferring it to the frequency domain, I would find the maximum frequency present for the set of data corresponding to each τ value. I would then test this maximum value to see which bass guitar note it was closest to and define another Shannon filter essentially filtering this note out. Unfortunately, my computer would crash whenever this code was run so I turned to a new idea.

My next attempt started out similarly to the first, except instead of determining the note that the maximum frequency was closest to, I used the maximum frequency as the center for a new Gaussian filter. This Gaussian filter was created in the frequency domain and then added to itself a couple of times to

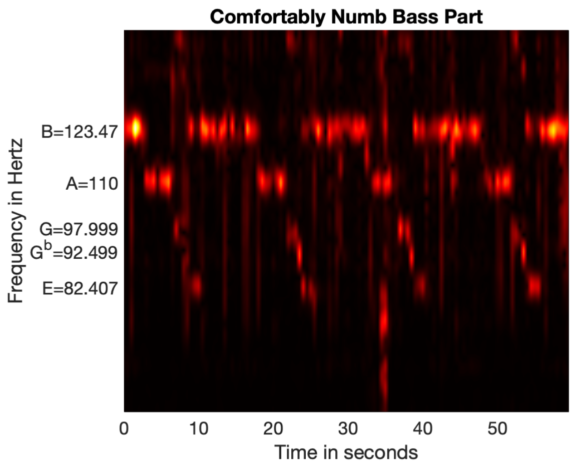

Figure 2: Spectrogram of the bass guitar's music score for the song "Comfortably Numb" by Pink Floyd

Algorithm 2: Reproduce Music Score Through Use of a Shannon Filter

```
Extract data from song file
Define vector tau, window centers, and a, window size
Perform fft on sample data array
Define Shannon window function to determine indices
Set values in array corresponding to high frequencies to 0
for j = 1 : length(tau) do
    Define Gaussian window function
    Multiply filter by sample data array, element-wise
    Perform fft on array
    fftshift the absolute value of the array
    Insert array into column j of a matrix
end for
Create spectrogram of results with notes labeled
```

account for overtones. Then, it would be shifted and multiplied element-wise with the sample frequency data (which had already been multiplied by a different Gaussian window filter before the maximum frequency was determined). Unfortunately, this code resulted in a spectrogram of only frequencies equal to zero. Perhaps the second Gaussian filter (which was actually the third filter used on the data, after the Shannon filter and the first Gaussian filter) filtered too much of the frequency data out.

Finally, I decided to avoid filtering by the maximum frequency and instead define a Shannon filter that restricted the domain. At first I tried creating a single Shannon filter that filtered out all data below 250 Hertz and above 750 Hertz, leaving only frequencies that were in the range of the guitar from "Sweet Child O' Mine." To do this, I multiplied a Shannon filter, similar to what I had used for the bass guitar part of "Comfortably Numb" except with for a different value in Hertz, by another Shannon filter. This resulted in an error. I then tried to create two separate Shannon filters, one filtering out all data above 750 Hertz and another for all data below 250 Hertz, but this resulted in a spectrogram with all zero frequencies.

Finally, I decided to create a Shannon filter that filtered out all frequencies above 750 Hertz through setting the corresponding indices in the sample frequency data to be zero. The data was then transformed back into the time domain. A similar for loop to what was defined previously was created, where a Gaussian filter function was applied to the sample frequency data. This data was then put into the frequency domain and its absolute value was `fftshift`-ed and added to a matrix. This data was visualized through a spectrogram whose y-axis limits were defined to be the frequencies of the guitar from the Guns 'n Roses song. All of the code used for this assignment in located in Appendix B.

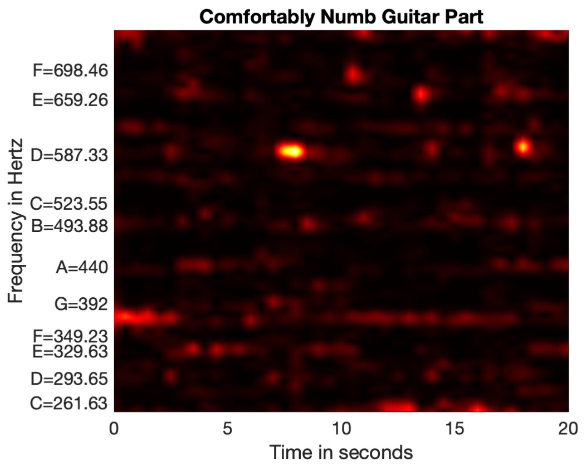

Figure 3: Spectrogram of the reproduction of the guitar solo in the song "Comfortably Numb" by Pink Floyd

4 Computational Results

As aforementioned, the goal of this assignment was to recreate the music scores of "Sweet Child O' Mine" and "Comfortably Numb." After the Gabor transform was completed and the values of the frequency components and rows of frequency data for "Sweet Child O' Mine" were restricted accordingly, the spectrogram depicted in Figure 1 was produced. At each time-step, the lowest bright spot on the spectrogram corresponds to the primary note being played by the guitar. Any lighter spots above this are overtones. The x-axis represents the time in seconds and the y-axis represents the frequency in Hertz. The notes corresponding to the frequencies are also labeled.

Figure 2 shows the spectrogram of the bass guitar's music score for "Comfortably Numb" by Pink Floyd. It can be read using the same logic as was used for Figure 1. This particular visual was what was produced through the utilization of the Shannon filter discussed in Section 3.

Figure 3 displays the recreation of the guitar solo in "Comfortably Numb." The range of output values were restricted to the approximate range of the guitar part in "Sweet Child O' Mine." Since the recovery of the guitar solo is not perfect, all notes between these frequencies are labeled.

5 Summary and Conclusions

There are many different filtering methods that can be used for signal processing. Each comes with associated pros and cons. In this paper, we considered the Gabor transform which uses a filter function, often set to be a Gaussian. It sweeps across different regions of the time domain, analyzing the time and frequency information at each window of time. The Gabor transform is a compromise between obtaining time domain and frequency domain information, as opposed to the Fourier transform, which only keeps frequency domain information.

After processing corresponding audio files, the music score for the guitar part for the Guns 'n Roses song "Sweet Child O' Mine" and the bass part for the Pink Floyd song "Comfortably Numb" were recreated. The results were plotted in spectrograms, which included time information and frequency information labeled by the note and the frequency in Hertz. Originally, after being put through a Gabor transform, the data was filtered by restricting the domain of the frequency data outputs to the appropriate range. In the next step, the bass guitar part of "Comfortably Numb" was first filtered using a Shannon filter in frequency space and then put through the Gabor transform. Finally, a Shannon filter and a restriction on the y-axis limits, in addition to the Gabor transform, were utilized to create a spectrogram representing the guitar solo in "Comfortably Numb."

Appendix A MATLAB Functions

- `player = audioplayer(Y,Fs)` accepts sample data `Y` and a sample rate `Fs` and returns an audio player called `player` for it.
- `[Y,Fs] = audioread(filename)` accepts an audio file and generates sample data, `Y`, in the form of a vector and a sample rate for that data, `Fs`, as a scalar.
- `X = ifft(Y)` uses the fast Fourier transform algorithm to return the inverse Fourier transform of an inputted array `Y`.
- `k = find(X,n)` accepts an array `X`, which can be specified with a condition such as `X>10`, and returns indices that correspond to the first `n` values that satisfy the given condition. (The default condition, if only `X` is entered, is to look for elements that are nonzero.)
- `Y = fft(X)` uses the fast Fourier transform algorithm to return the Fourier transform of an inputted array `X`.
- `Y = fftshift(X)` takes an array `X` in the frequency domain and returns a shifted version of it, such that the element corresponding to the zero-frequency is located in the middle of the array.
- `playblocking(playerObj)` plays complete audio file corresponding to an audioplayer object called `playerObj`
- `pcolor(X,Y,C)` returns a psuedocolor plot of the values in array `C`, where `X` and `Y` give the $x-$ and $y-$coordinates corresponding to the vertices of the plotted region.
- `yticklabels(labels)` sets the displayed labels on the y-axis at the locations specified by `yticks(ticks)`, using a string array called `labels`.
- `yticks(ticks)` sets the values on the y-axis where the tick marks should appear, using a vector of increasing values called `ticks`.

Appendix B MATLAB Code

```matlab
clear; close all; clc

% Part 1

% Plot and gather data about GNR signal
figure
[y1, Fs1] = audioread('GNR.m4a');
gnrTime = length(y1)/Fs1; % record time in seconds, size of domain
plot((1:length(y1)),y1);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Sweet Child O Mine');
p8 = audioplayer(y1,Fs1); playblocking(p8);

% Define necessary variables
L1 = gnrTime; % spatial domain, length of recording
n1 = length(y1); % Fourier modes, length of vector
t21 = linspace(0,L1,n1+1); % time discretization
t1 = t21(1:n1); % take first n points only
t1 = t1';
k1 = (1/L1)*[0:(n1/2 - 1) -n1/2:-1]; % leave out 2pi to stay in Hertz
ks1 = fftshift(k1); % frequency components
```

```matlab

% Define Gabor Transform
tau1 = linspace(0,L1,101); % center of window
a1 = 1500; % window size

for j = 1:length(tau1)
   g1 = exp(-a1*(t1-tau1(j)).^2); % Gaussian window function
   gnrFiltered = g1.*y1;
   FgnrFiltered = fft(gnrFiltered);
   FgnrFilteredSpec(:,j) = fftshift(abs(FgnrFiltered));
end

% Restrict domain of displayed values
idx11 = find(ks1 > 75,1); % first time ks1 value > 75
idx21 = find(ks1 > 800,1); % first time ks1 value > 800
ks1lim = ks1(idx11:idx21);
Fgnrlim = FgnrFilteredSpec(idx11:idx21,:);

% Create spectrogram
figure
pcolor(tau1,ks1lim,Fgnrlim)
set(gca,'Fontsize',16)
colormap(hot)
xlabel('Time in seconds'), ylabel('Frequency in Hertz')
shading interp
yticks([277.18 369.99 415.30 554.37 698.46 739.99])
yticklabels({'D^b=277.18','G^b=399.99','A^b=415.30','D^b=554.37','F=698.46','G
   ^b=739.99'})
title('Sweet Child O Mine Guitar Part')

% Plot and gather data about Floyd signal
figure
[y2, Fs2] = audioread('Floyd.m4a');
floydTime = length(y2)/Fs2; % record time in seconds, size of domain
plot((1:length(y2)),y2);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Comfortably Numb');
p9 = audioplayer(y2,Fs2); playblocking(p9);

% Define necessary variables
L2 = floydTime; % spatial domain, length of recording
n2 = length(y2); % Fourier modes, length of vector
t22 = linspace(0,L2,n2+1); % time discretization
t2 = t22(1:n2); % take first n points only
t2 = t2';
k2 = (1/L2)*[0:((n2+1)/2 - 1) (-n2+1)/2:-1]; % leave out 2pi to stay in Hertz
ks2 = fftshift(k2); % frequency components

% Define Gabor Transform
tau2 = 0:0.5:L2; % center of window
a2 = 100; % window size

for k = 1:length(tau2)
   g2 = exp(-a2*(t2-tau2(k)).^2); % Gaussian window function
```

```
    floydFiltered = g2.*y2;
    FfloydFiltered = fft(floydFiltered);
    FfloydFilteredSpec(:,k) = fftshift(abs(FfloydFiltered));
end

% Restrict domain of displayed values
idx12 = find(ks2 > 50,1); % first time ks1 value > 50
idx22 = find(ks2 > 150,1); % first time ks1 value > 150
ks2lim = ks2(idx12:idx22);
Ffloydlim = FfloydFilteredSpec(idx12:idx22,:);

% Create spectrogram
figure
pcolor(tau2,ks2lim,Ffloydlim)
set(gca,'Fontsize',16)
colormap(hot)
xlabel('Time in seconds'), ylabel('Frequency in Hertz')
shading interp
yticks([82.407 92.499 97.999 110 123.47])
yticklabels({'E=82.407','G^b=92.499','G=97.999','A=110','B=123.47'})
title('Comfortably Numb Bass Part')

% Part 2

tau2 = 0:0.5:L2; % center of window
a2 = 100; % window size

% Only keep frequency information for frequencies below 200
floydF=fft(y2);
s2 = abs(ks2)<150; % Shannon window function
floydF(s2) = 0;
TfloydSFiltered = ifft(floydF);

% Perform Gabor Transform
for k = 1:length(tau2)
    g2 = exp(-a2*(t2-tau2(k)).^2); % Gaussian window function
    NewFloydFiltered = g2.*TfloydSFiltered;
    NewFfloydFiltered = fft(NewFloydFiltered);
    NewFfloydFilteredSpec(:,k) = fftshift(abs(NewFfloydFiltered));
end

% Create spectrogram
figure
pcolor(tau2,ks2,NewFfloydFilteredSpec)
set(gca,'ylim',[50 150],'Fontsize',16)
colormap(hot)
xlabel('Time in seconds'), ylabel('Frequency in Hertz')
shading interp
yticks([82.407 92.499 97.999 110 123.47])
yticklabels({'E=82.407','G^b=92.499','G=97.999','A=110','B=123.47'})
title('Comfortably Numb Bass Part')

% Part 3

```

```
tau2 = 0:0.5:20; % center of window
a2 = 900; % window size

% Only keep frequency information for frequencies below 200
y2f=fft(y2);
s21 = abs(ks2) < 750; % Shannon window function
y2f(s21) = 0;
TfloydSFiltered = ifft(y2f);

% Perform Gabor Transform
for k = 1:length(tau2)
   g2 = exp(-a2*(t2-tau2(k)).^2); % Gaussian window function
   NewFloydFiltered = g2.*TfloydSFiltered;
   NewFfloydFiltered = fft(NewFloydFiltered);
   NewFfloydFilteredSpec(:,k) = fftshift(abs(NewFfloydFiltered));
end

% Create spectrogram
figure
pcolor(tau2,ks2,NewFfloydFilteredSpec)
set(gca,'ylim',[250 750],'Fontsize',16)
colormap(hot)
xlabel('Time in seconds'), ylabel('Frequency in Hertz')
shading interp
yticks([261.63 293.65 329.63 349.23 392.00 440.00 493.88 523.55 587.33 659.26
    698.46])
yticklabels({'C=261.63','D=293.65','E=329.63','F=349.23','G=392','A=440',...
     'B=493.88','C=523.55','D=587.33','E=659.26','F=698.46'})
title('Comfortably Numb Guitar Part')
```

Listing 1: This is the code used for reproducing the guitar and bass parts for every part of this project.