Guitar and Bass Audio Visualization

Avery Milandin

Due February 9th, 2021

Abstract

In this Project we generate something similar to guitar score sheets from guitar music in audio files. Using the Gabor transform, we are able to generate plots of which frequencies are present at which times in the songs and hence reproduce a score sheet. We also use filtering to isolate bass from guitar.

1 Introduction and Overview

Our data is given to us in .m4a files, and we are able to read these files into MATLAB as arrays representing audio signals over time. Our goal is to reproduce the guitar score for the GNR.m4a file, the bass for the Floyd.m4a file, and the guitar for the Floyd.m4a file. By taking a fourier transform of these signals we can see how much of each frequency makes up the audio from a file, and since we know the frequency range of guitar and bass, we can apply a band pass filter to our signal in frequency space to remove any frequencies that are not within the range of our instruments. After filtering, we can transform back to the time domain where we can apply a Gabor transform to our filtered signals in order to generate a plot of which frequencies are present at which times. We call this plot a spectrogram.

2 Theoretical Background

Perhaps the most important theoretical concept to understand for this project is the Gabor transform. The Gabor transform is very similar to the Fourier transform, and the formula is as follows:

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

Note that the only difference between this formula and the regular Fourier transform is the $g(t-\tau)$ function that the original function is multiplied by. This function is usually a filter or "window function" centered at τ. This means that our original function stays the same, except values that are not near τ are removed. Plotting $\tilde{f}_g(\tau, k)$ allows us to see which frequencies are present at the times spanned by the window function. For this project, we use a discrete version of this transform, and our window function is a Gaussian that takes an additional parameter, a, which can be used to make the window of our window function wider or narrower.

This brings us to the Heisenberg Uncertainty Principle. Notice that with this approach we do not get any information about what frequencies are present at different times within the window spanned by the window function. If we try making our window extremely small to solve this, then the window will be too small for the transform to detect anything other than extremely high frequencies. If we make the window too large, on the other hand, we will know all of the frequencies that are present, but we will have very little information about when they are present. This is the idea of the Heisenberg Uncertainty Principle; it states that we cannot know everything about time and frequency at the same time, so we need to choose a window that gives us useful information about both time and frequency, but it will never be perfect.

3 Algorithm Implementation and Development

Before developing any algorithms, we first need to define the variables that we will use for most of the algorithms. We will define the signal, time domain, frequency domain, and the filter. The filter needs to filter out all angular frequencies less than -4000 or greater than 4000 because frequencies outside of this range are not able to be produced by a guitar, so we know that they are just noise in our signal. Note that the only difference between angular frequency and frequency is that angular frequency has been scaled by 2π. The following code does this for the GNR.m4a audio file.

```
[y, Fs] = audioread('GNR.m4a');
S = y'; %signal
t = (1:length(y))/Fs; %times
n = length(t);
L = t(end); %length of time domain
k = (2*pi/L)*[0:n/2 -n/2:-1]; %frequency domain
ks = fftshift(k);
fil = zeros(1,length(ks)); %band pass filter
for i = 1:length(fil)
    if (abs(ks(i))>4000)
        fil(i) = 0;
    else
        fil(i) = 1;
    end
end
```

Note that the condition on the "if" statement will be different for the Floyd.m4a file. For the bass in that file, the filter will filter out all angular frequencies greater than 1000, and for the guitar the filter will filter out all frequencies greater than 4000 or less than 1000 in order to isolate the guitar from the bass.

Now that we have the variables we need, we can implement the first algorithm, which is filtering our signal with the band-pass filter. Note that we could just apply this filter to the spectrogram that we will produce later, but by doing this now we get a better visualization of what the filter is doing. See figures 1, 2, and 3 to see which frequencies of our signals we filter out. The steps are as follows:

Algorithm 1

1. Take the fast Fourier transform of our signal.
2. Multiply our signal by our band-pass filter to remove unwanted frequencies from our signal.
3. Take the inverse fast Fourier transform of the filtered signal to give us the filtered signal in the time domain

The code for this is as follows:

```
St = fft(S);
Stf = fftshift(St).*fil;
Sf = ifft(fftshift(Stf));
```

Now that we have our filtered signal, Sf, we can create a spectrogram, which shows us the score for the guitar. The steps are as follows:

Algorithm 2

1. Define parameters a and τ for our window function. τ is a vector because we will be taking Gabor transforms in a loop in order to slide the window function across our signal at each time step. These parameters are determined by guessing and then revising guesses later based on results.
2. Populate a matrix with zeros that is the length of our signal by the length of τ. This will be for plotting the spectrogram later.
3. Loop over the tau vector and perform steps 4 through 6 on every iteration.

4. Define the window function as a gaussian with parameters a, which controls the width, and τ, where τ is the current value that is being looped over in the τ vector and determines where the window function is centered.

5. Multiply our window function by our filtered signal.

6. Take the fast Fourier transform of our windowed and filtered signal, and save the result as a column of the pre-constructed matrix.

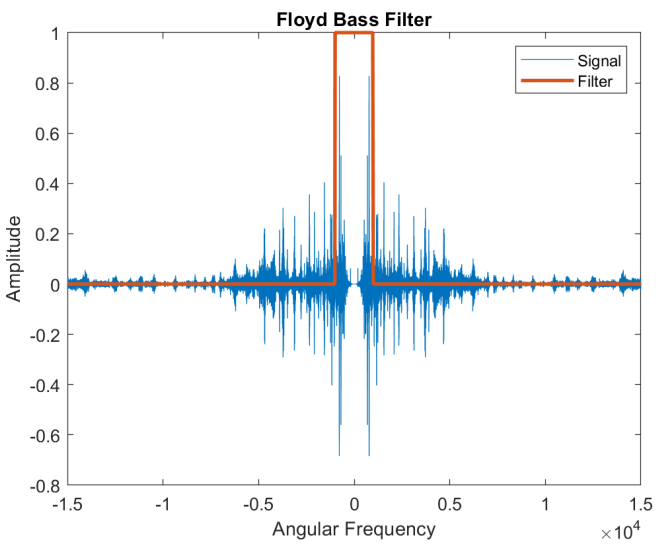
7. Using the matrix produced during the loop, create a shaded plot. The x-axis will be time, and the y-axis will be frequency in Hz. Our frequency data in our matrix is in angular frequency, so we divide it by 2π to get Hz. Shade points on the plot with a bright color if they correspond to frequencies present with a high amplitude for the given times. We do this step using the pcolor command; see appendix A.

The code for this algorithm is as follows:

```
a = 70;
tau = 0:0.2:L;
Sgtf_spec = zeros(length(S),length(tau));
for j = 1:length(tau)
   g = exp(-a*(t - tau(j)).^2); % Window function
   Sgf = g.*Sf;
   Sgtf = fft(Sgf);
   Sgtf_spec(:,j) = fftshift(abs(Sgtf));
end

figure(3)
pcolor(tau,ks/(2*pi),Sgtf_spec)
shading interp
set(gca,'ylim',[0 600],'Fontsize',16)
colormap(hot)
colorbar
xlabel('Time (t)'), ylabel('Frequency (Hz)')
title("GNR Spectrogram")
```

4 Computational Results



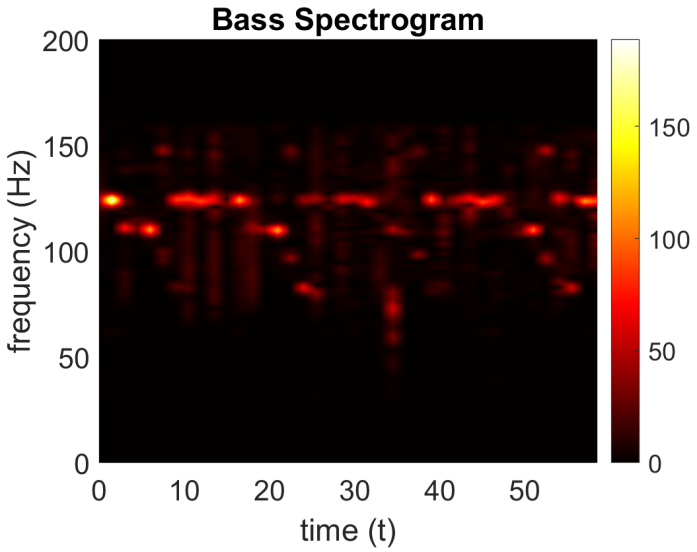
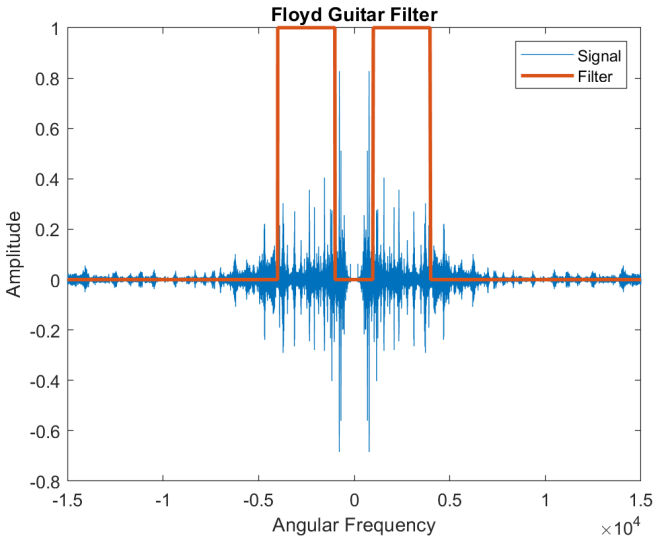


Figure 1: Here we can see that in order to isolate bass we need to filter out most of the frequencies in our signal.



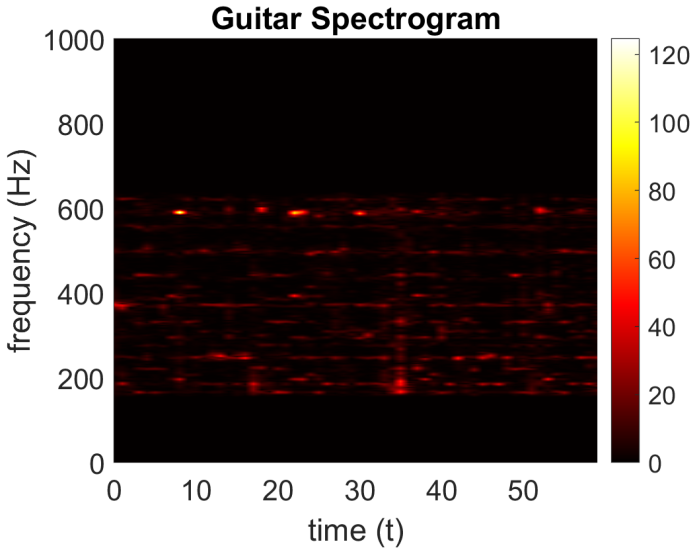


Figure 2: Here we can see that after filtering out overtones and bass, we did not get a very clean spectrogram. This is to be expected for though because of how many notes are played by the guitar in this song.

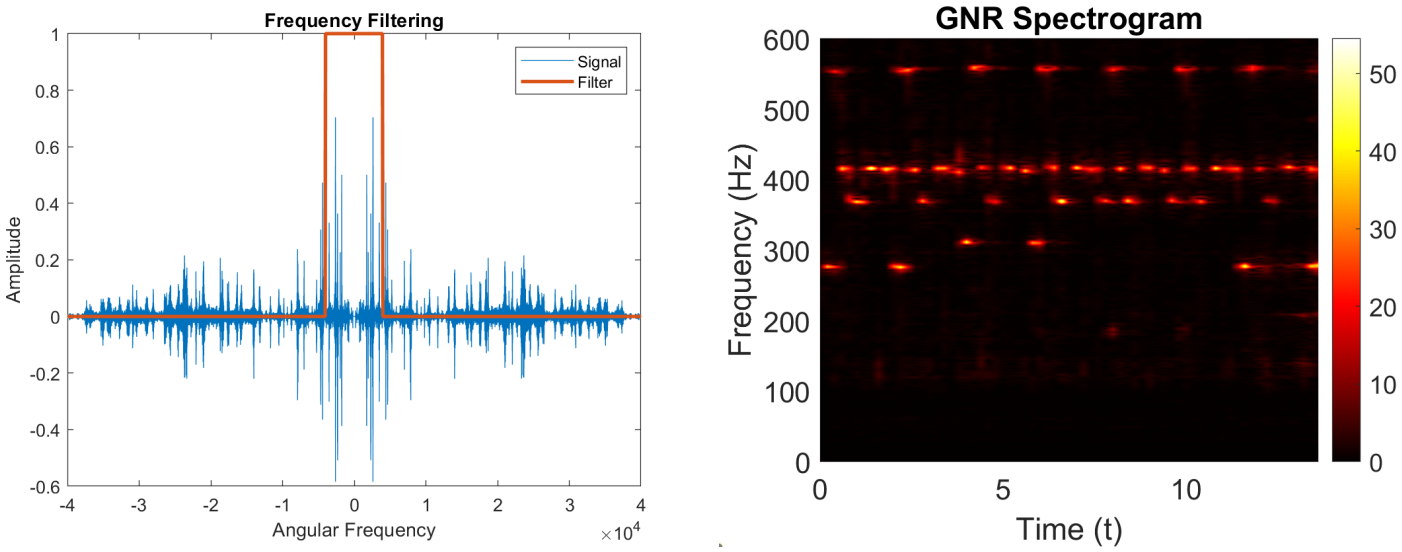


Figure 3: Here we can see that filtering out overtones led to a very clean spectrogram of our signal.

5 Summary and Conclusions

For this project we used the Gabor transform to process audio data representing songs in order to develop spectrograms similar to score sheets for different instruments present in the songs. The only difference is that our plots have frequency in Hz on the y-axis rather than labeled notes. We developed fairly clean spectrograms for the guitar in GNR.m4a and for the bass in Floyd.m4a. For the guitar in Floyd.m4a, the spectrogram does not look nearly as clean, but it can be seen that there are still distinguishable highlighted sections that last for a clear amount of time. This is to be expected since the Floyd.m4a clip is much longer than the GNR.m4a clip, and one can tell by listening to the audio that there are a lot more guitar notes played per unit of time in the Floyd clip than the GNR clip, so the spectrogram probably would not have looked perfect no matter what.

Appendix A MATLAB Functions

- `[y, Fs] = audioread('Floyd.m4a')` takes an audio file as input and returns an array representing the signal, y, as well as the rate at which audio points are played in Hz, Fs
- `ks = fftshift(k)` takes an array as input and returns a new array with the first half and second half swapped. This is needed since when MATLAB performs a Fourier Transform it returns an array with each half of the array on the opposite side that we would expect it on.
- `St = fft(S);` takes an array as input and returns the fast Fourier transformed array.
- `pcolor(X,Y,Z)` takes vectors X and Y as input and generates a shaded plot with axes matching X and Y vectors. Z is a matrix with dimensions X by Y, and each point represents a point on the grid defined by X and Y. The greater a value in the Z matrix, the more brightly colored the corresponding pixel is on the pcolor plot.

Appendix B MATLAB Code

File 1 of 2

```
clear all; close all;

figure(1)
[y, Fs] = audioread('Floyd.m4a');
tr_gnr = length(y)/Fs; % record time in seconds
plot((1:length(y))/Fs,y);
xlabel('Time [sec]'); ylabel('Amplitude');
title("Floyd");
p8 = audioplayer(y,Fs); playblocking(p8);

%% define variables
S = y'; %signal
t = (1:length(y))/Fs; %times
n = length(t);
L = t(end); %length of time domain
k = (2*pi/L)*[0:n/2 -n/2:-1]; %frequency domain
ks = fftshift(k);
fil = zeros(1,length(ks)); %band pass filter
for i = 1:length(fil)
    if (abs(ks(i))>1000)
        fil(i) = 0;
    else
        fil(i) = 1;
    end
end


%% Filtering out everything but bass
St = fft(S);
Stf = fftshift(St).*fil;
Sf = ifft(fftshift(Stf));
plot(ks,fftshift(St)/max(abs(St)))
xlim([-15000 15000])
hold on
plot(ks, fil, "LineWidth", 2)
xlabel("Angular Frequency")
ylabel("Amplitude")
title("Floyd Bass Filter")
legend("Signal", "Filter")


%% creating spectrogram for bass

a = 70;
tau = 0:1.5:L;
Sgtf_spec = zeros(length(S),length(tau));
for j = 1:length(tau)
   g = exp(-a*(t - tau(j)).^2); % Window function
   Sgf = g.*Sf;
   Sgtf = fft(Sgf);
   Sgtf_spec(:,j) = fftshift(abs(Sgtf));
```

```
end

figure(2)
pcolor(tau,ks/(2*pi),Sgtf_spec)
shading interp
set(gca,'ylim',[0 200],'Fontsize',16)
colorbar
colormap(hot)
title("Bass Spectrogram")
xlabel('time (t)'), ylabel('frequency (Hz)')
%% filtering out everything but guitar
fil = zeros(1,length(ks));
for i = 1:length(fil)
    if (abs(ks(i))<1000 | abs(ks(i))>4000)
        fil(i) = 0;
    else
        fil(i) = 1;
    end
end
Stf = fftshift(St).*fil;
plot(ks,fftshift(St)/max(abs(St)))
xlim([-15000 15000])
hold on
plot(ks, fil, "LineWidth", 2)
xlabel("Angular Frequency")
ylabel("Amplitude")
title("Floyd Guitar Filter")
legend("Signal", "Filter")
Sf = ifft(fftshift(Stf));
%% creating spectrogram for guitar
a = 140;
tau = 0:1:L;
Sgtf_spec = zeros(length(S),length(tau));
for j = 1:length(tau)
   g = exp(-a*(t - tau(j)).^2); % Window function
   Sgf = g.*Sf;
   Sgtf = fft(Sgf);
   Sgtf_spec(:,j) = fftshift(abs(Sgtf)); % We don't want to scale it
end

figure(3)
pcolor(tau,ks/(2*pi),Sgtf_spec)
shading interp
colormap(hot)
set(gca,'ylim',[0 1000],'Fontsize',16)
title("Guitar Spectrogram")
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')
```

File 2 of 2

```
clear all; close all;

figure(1)
[y, Fs] = audioread('GNR.m4a');
```

```
tr_gnr = length(y)/Fs; % record time in seconds
plot((1:length(y))/Fs,y);
xlabel('Time [sec]'); ylabel('Amplitude');
title("GNR");
p8 = audioplayer(y,Fs); %playblocking(p8);
%% define variables
S = y';
t = (1:length(y))/Fs;
n = length(t);
L = t(end);
k = (2*pi/L)*[0:n/2-1 -n/2:-1];
ks = fftshift(k);
fil = zeros(1,length(ks));
for i = 1:length(fil)
    if (abs(ks(i))>4000)
        fil(i) = 0;
    else
        fil(i) = 1;
    end
end


%% Filtering
St = fft(S);
Stf = fftshift(St).*fil;
Sf = ifft(fftshift(Stf));
figure(2)
plot(ks,fftshift(St)/max(abs(St)))
xlim([-40000 40000])
hold on
plot(ks, fil, "LineWidth", 2)
xlabel("Angular Frequency")
ylabel("Amplitude")
title("Frequency Filtering")
legend("Signal", "Filter")


%% creating spectrogram

a = 70;
tau = 0:0.2:L;
Sgtf_spec = zeros(length(S),length(tau));
for j = 1:length(tau)
   g = exp(-a*(t - tau(j)).^2); % Window function
   Sgf = g.*Sf;
   Sgtf = fft(Sgf);
   Sgtf_spec(:,j) = fftshift(abs(Sgtf));
end

figure(3)
pcolor(tau,ks/(2*pi),Sgtf_spec)
shading interp
set(gca,'ylim',[0 600],'Fontsize',16)
colormap(hot)
```

```
colorbar
xlabel('Time (t)'), ylabel('Frequency (Hz)')
title("GNR Spectrogram")
```