Amath 482 Homework 2

THE USE OF GABOR TRANSFORMS IN ANALYZING SIGNALS

MICHAEL GABALIS

Abstract: This paper aims to demonstrate the usefulness of the Gabor Transform in extracting frequency information from time-varying signals (specifically audio signals). As many signals vary throughout time, this technique has a wide range of applications including computational neuroscience, image analysis, and the decomposition of audio files. This paper will discuss the downfalls associated to the Gabor transform and how some of those trade-offs can be mitigated using a sampling window. As a case study, we will use this technique to study a guitar solo from GNR's "Sweet Child O' Mine" and Pink Floyd's "Comfortably Numb".

University of Washington

February 2021

1 INTRODUCTION AND OVERVIEW

Many of the signals that we try to solve are time varying. Often the relative order of these frequencies carries information in and of itself and the time information of the local frequency content is important. When an audio files time information changes, we suddenly lose all meaning. Hence, when we apply the Fourier Transform to the global frequency content, we lose all time information that comes with that data. However, the Gabor Transform can be used to improve this short-coming of the Fourier Transform. By only looking at a specific window of time, we can break down the signal into very small increments, separating the frequency information for small portions of time. This means that we mitigate our loss of time information, although we can also lose some frequency resolution.

Another common theme in frequency filtering is the filtering of high or low frequencies, also called a low-pass filter and a high-pass filter, respectively. Often there is frequency information we know should not be there, and we can use these filters to take out some of the frequencies above or below a certain threshold (or only accept certain frequencies in a threshold).

In this paper, we use both techniques to decompose several pieces of music. The first is Gun's and Roses "Sweet Child O' Mine" and the second is "Comfortably Numb".

2 THEORETICAL BACKGROUND

2.1 FOURIER TRANSFORM

A Fast Fourier Transform (FFT) is an algorithm that computes the Discrete Fourier Transform (DFT) of a sequence. In this scenario, the Fourier Transform converts a signal from the spatial domain into the frequency domain and the inverse Fourier Transform converts the signal back into the spatial domain. The DFT is defined as:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi ikn/N}\ , k = 0, \dots , N-1 \qquad 1$$

The IFFT Transform is defined as:

$$X_n = \frac{1}{N} \sum_{k=0}^{N-1} x_k e^{2\pi ikn/N}\ , k = 0, \dots , N-1 \qquad 2$$

N is some integer 2^n and the Fourier Transform runs at O(Nlog(N)), meaning that it is a very efficient solver, one of the reasons we use it for filtering signals. k represents the wavenumber.

2.2 AVERAGING

Let's say our system has zero-mean noise in the frequency domain. That means we have some true signal mixed in with some random noise. Thus, at any given time our signal is the sum of the true signal plus the random noise. However, by Law of Large Numbers, as we increase the sample size we can approximate the signal down to the true signal. Therefore, if we have multiple realizations of the data with similar frequencies, the uncertainty in our signal can be reduced by taking the average in the frequency domain. Doing this, we can approximate the central frequency.

2.3 GABOR TRANSFORM

As said before, the downside to this technique is that it takes away from some of the spatial and temporal information. When averaging across multiple realizations of data we lose temporal information about the individual realizations of the data. This is especially an issue if the spatial information (audio signal) changes across time. Using a Gabor Transform is a compromise to limit the loss of temporal information without distorting the audio signal:

$$G[f](t,w) = \int_{-\infty}^{\infty} f(\tau)g(\tau - t)e^{-i\omega t}d\tau \qquad 3$$

Where g$(\tau - t)$ defines a window centered at time t. This window acts as a filter, allowing us to focus our frequency analysis on a specific window in time. As we move t across the temporal domain, we can get the corresponding frequencies at each point in time. This gives us the information on where in time each of these frequencies are occurring and we can detect the width of the window limits of the wavelengths of frequencies, thus we lose frequency resolution. However, expand the window and we begin to lose temporal resolution, so there is a constant trade-off between spatial and frequency resolution as dictated by the Heisenburg Uncertainty Principle. As a result of this trade-off, it is important to test different windows to find the appropriate one. The width of the window should be chosen based on the range of frequencies of interest because we want them all to fit inside the window. Some of the common Gabor Transforms are listed below:

$$\boldsymbol{Gaussian}: g(\tau - t) = e^{-a(\tau-t)^2} \qquad 4$$
$$\boldsymbol{Mexican\ Hat\ Wavelet}: g(\tau - t) = (1-(\tau-t)^2)e^{-\frac{\boldsymbol{a(\tau-t)^2}}{\boldsymbol{2}}}$$

$$\boldsymbol{Shannon}: g(\tau - t) = \begin{cases} 1 & x \epsilon \, [t - \frac{1}{2a}, t + \frac{1}{2a}] \\ 0 & otherwise \end{cases}$$

These windows decay at different rates and have their own specific properties and uses, so each can be chosen to fit the signal at hand.

2.4 High-pass and low-pass filtering

While these filters are great at filtering out noise and focusing on the information needed, sometimes we need to create a filter that blocks out not only noise, but certain ranges of frequencies as well. When we want to remove high frequency information, we can implement a low-pass filter that only keeps the data in below a certain threshold. For removing low frequency information, we implement a high-pass filter. We use band-pass filters to only keep a range of filters between two values. We can do this by creating filters that look like the ones above. However, I used a rectangular filter that looks like the Shannon filter so that I could create a hard stop between the filters I wanted and the ones I did not.

2.5 Spectrograms

After filtering through the frequency data, we can use a spectrogram to create a visualization of the results. A spectrogram plots the time on the x-axis, the frequencies on the y-axis, and the strength or amplitude of these frequencies is plotted as a color bar. The color intensity highlights the prevalence of a given frequency at a specific time. These spectrograms can be used to analyze the data at hand and pick up patterns that we can't see when looking at a much of data.

3 Algorithm Implementation and Development

3.1 Fast Fourier Transform Implementation

The primary tool we using for converting between the spatial and frequency domain will be the FFT and the IFFT. The spatial and spectral resolution of our data is given to be ($L = 10, n = 443392$, so we begin the code by defining the spectral and signal domains. As FFT assumes the domain to be [-π,π], and not [-L,L], we have to rescale our wavenumbers k by $\frac{2\pi}{L}$ and shift them by defining $ks = fftshift(k)$ for convenience. However, the scaling of our wavenumbers isn't necessarily needed in our case because we have an audio file which is already in Hz.

3.2 Analysis of Gabor Windows: Sweet Child O' Mine

We begin by loading a sample of Gun's and Roses song "Sweet Child O' Mine" as a simple vector of the amplitudes of the frequencies. Using the sampling rate Fs, we can reconstruct the time domain of the signal. We rename the vector *y* to *S* and transpose it so that we can multiply it more easily (without having to transpose every single time). Given the time domain we found, we reconstruct the wavenumbers *k*, shifting them using the *fft* command to be centered at zero (Appendix B, code 1-8). From there we analyze the data given to us by plotting the raw data of the frequencies and the shifted Fourier Transform of our data to get a better picture of our data and a rough estimate for how we should set up the Gabor Transform (Appendix B, code 34-51). Our goal is to use the Gabor Transform to analyze the frequency content of this music as a function of time. First, we define some of the properties of the window: the sampling rate *tstep* (how much it slides each iteration), the window width which is inversely related to *a,* and the Gabor function that we decide to use from **equation 4** (Appendix B, code 61-64). For each sampling frame in our time domain, we apply the Gabor filter to the signal in the time domain, use the Fast Fourier Transform to convert to the frequency domain, dividing by two pi to get back into our original domain. Using a max function, we can find the most probable frequencies of the notes that are being played. In each sampling frame, the resulting frequency information is stored for future use in a spectrogram (Appendix B, code 127-131). After obtaining all of the frequency information by going through each sampling frame, we have the information available to create a spectrogram (using *pcolor* to plot each sampling frame, the frequency numbers, and the amplitude of these frequencies) (Appendix B, code 133-146).

3.3 Choice of Window and Sampling Rate

In the above code we chose a random window width and sampling rate so that we could get a basis for what range of numbers we could use. However, we want to get a clearer spectrogram so we can have the best knowledge for what the notes will be and at what times. In other words, we want to have the clearest frequency information without filtering it out too much. To do this, we can create a for loop that loops through several different window widths and saves each of those graphs in a file (Appendix B, code 54-83). To find a sampling rate that gives an accurate representation without using too much processing power, we can loop through multiple sampling rates as well (Appendix B, code 86-115). Now that we have the best window and sampling rate, we can use the process from above to create the clearest spectrograms.

3.4 High and Low-Pass Filtering – Floyd's Comfortably Numb

When analyzing the song: Comfortably Numb, we can use similar techniques to create a spectrogram to find the notes of the song. However, because of the size of the song, it is almost impossible to filter the song well and still get a clear picture. Hence, the song can be cut down to only the first 10 seconds so that we can showcase how our filter works, without using too much processing power (Appendix B, code 9-19). From here we can filter the song the same way we did with "Sweet Child O' Mine", by finding the best window and sampling rate, using a

Gabor filter, then plotting the spectrogram (Appendix B, code 52-147). However, Comfortably Numb is different because there is more than one instrument playing. To filter out the notes played by one instrument we must use a low-pass filter for the bass, and a band-pass filter for the guitar. For the bass, we can basically use a rectangular filter that cuts off at a frequency above a certain point, *fftshift* the filter, and apply that to the Fast Fourier Transformed signal. (Appendix B, code 20-26). This will make the frequencies above that point equal to zero and leave everything else untouched. The way I found the cutoff frequency was by looking at the spectrogram of the entire song and knowing that the bass will be the lowest frequencies. For the guitar solo we can do a very similar process but make our rectangle function into a band-pass filter by cutting off frequencies above and below a certain threshold (Appendix B, code 28-33). These thresholds were again achieved by looking at the spectrogram of the entire song. After applying these low-pass or band-pass filters to the song, all that is left to do is apply these filters to each sampling rate and create the spectrogram. This process is the same as when we were filtering out the entire song or the Guns and Roses song (Appendix B, code 148-176).

3.5 Creating the Notes and Applying Them

Now that we have the spectrograms for *Sweet Child O' Mine* and the guitar and bass filtered for *Comfortably Numb,* all that is left to do is find the notes given by the spectrogram. To do this we can extract the max frequencies that were filtered out of these songs during each sampling rate frame. To give letters to these frequencies, we can create a vector of the notes playable in music (A-G) and a corresponding vector of the base frequencies of these notes. Using a *for loop*, we can find the frequencies of the notes at each octave (Appendix B, code 179-197). After we have a number for each note, we can use the *min* function to match our max frequencies we filtered out to an actual note. We now have a beautiful vector of notes that correspond to the frequencies played in each song. From here the *gtext* function can be used to paste the notes onto the spectrogram and give a representation of what notes are being played!

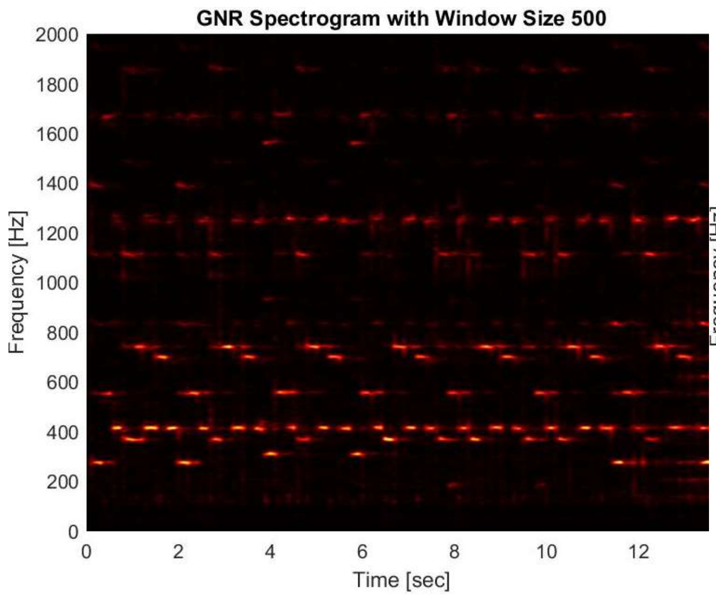

Figure 1a: Standard Spectrogram

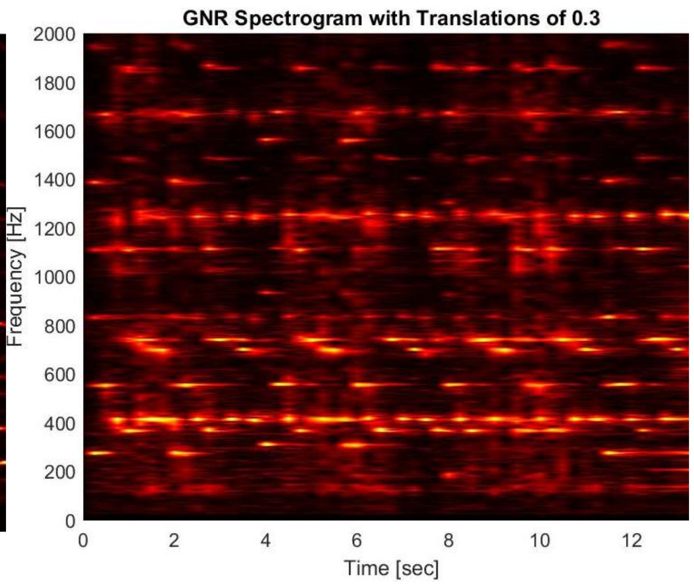

Figure 1b: Spectrogram with too low of sampling size

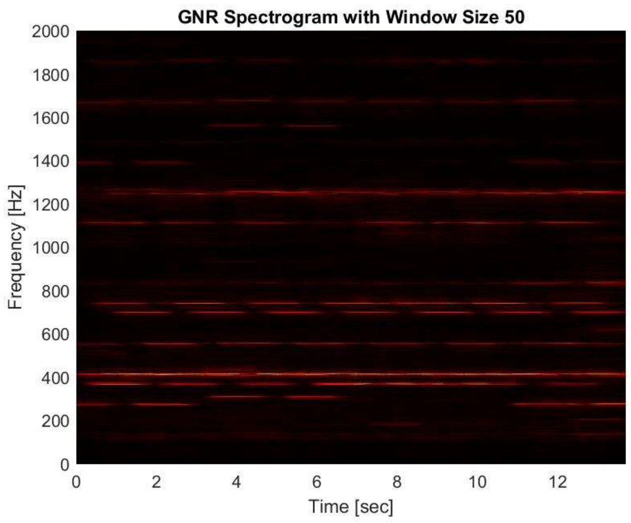

Figure 2c: Spectrogram with too large of a window

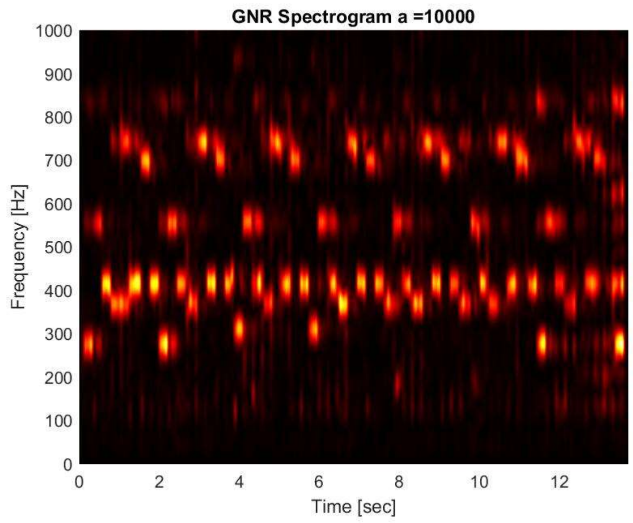

Figure 1d: Spectrogram with too small of a window

4 Computational Results

4.1 Analysis of Gabor Windows And Sweet Child O' Mine

We began by exploring how various properties of the Gabor window and the sampling size affect the outcome. As seen in **figure 1a,** we see what the perfect window and sampling size is for the *Sweet Child O' Mine* song. We will use this as a benchmark to compare what happens when you do not use the proper window or sampling size. We see that there are several bands of frequencies that correspond to given notes, and that will be observed later.

When we use too low of a sampling size as we did in **figure 1b**, we are looking at too much temporal information at one time, making the wavenumbers more noticeable, but at a loss of the temporal variation which is important in music. With a window that is too wide, such as **figure 1c,** we gain frequency resolution at the cost of temporal resolution. This allows us to look for a wide range of frequencies and makes the large bands more noticeable, but we lost the temporal information important to music. In **figure 1d,** we narrow the window to *a = 10000*, that gives a very good look of temporal variation. Normally we would lose track of our long wavelength signals, but here those frequencies are not prominent so that is not much of an issue. However, this reduction in frequency resolution makes it more difficult to differentiate long-term patterns and to determine individual frequencies because of the blurred areas of activity.

When using the music score that we created, we can apply the notes to the spectrogram in **figure 1a. Figure 2** shows the result of the spectrogram with notes. Going from lowest frequency to highest frequency we have C#, D#, F#, G#, C# (high), F (high), F# (high). In terms of frequencies in Hz this is 554, 622, 740, 831, 1109, 1397, and 1480. The data was sampled about every tenth of a second with a window of a = 250. It is clear that the Fast Fourier Transform and Gabor Transform was able to filter this data very well. Each note is precisely played with very little overtones visible. It seems that the lower notes come out much more clear than the higher notes, meaning that filtering through the overtones may make the higher frequencies less clear. We can also note that that all of these notes are being played very fast, meaning that we are going at a high bpm or are dealing with half notes. From this spectrogram, I pieced together a music scale until it sounded like the original song and had the frequencies we got. As

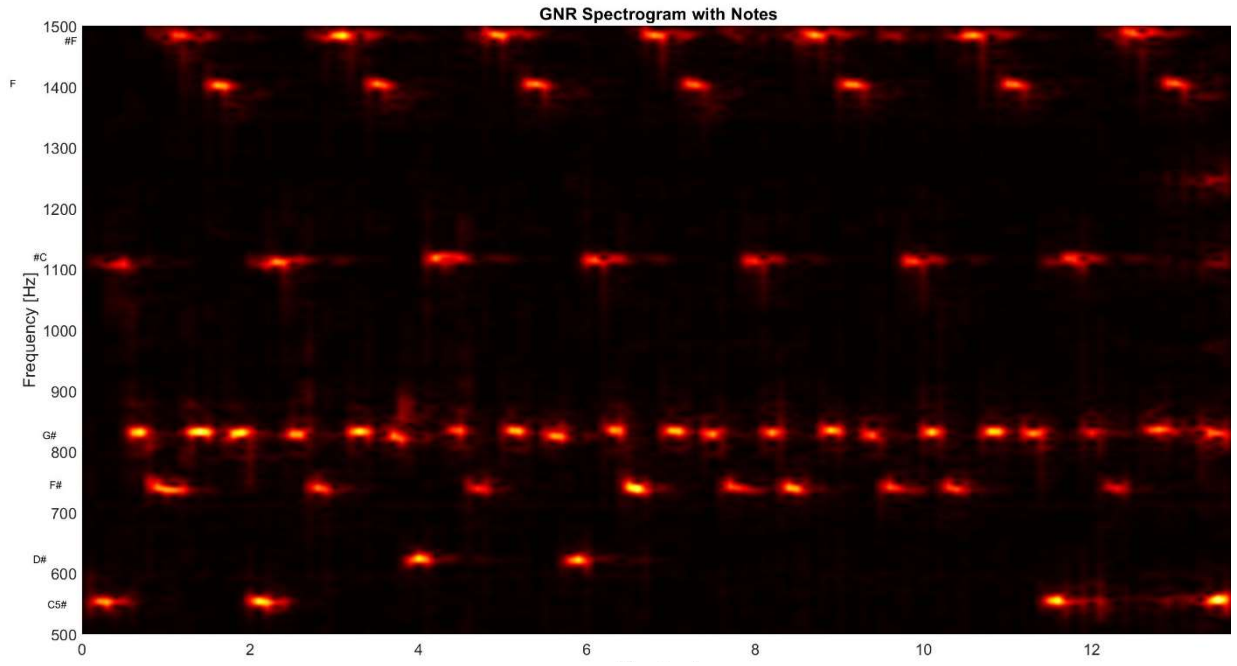

Figure 2: Sweet Child O' Mine Spectrogram with Notes

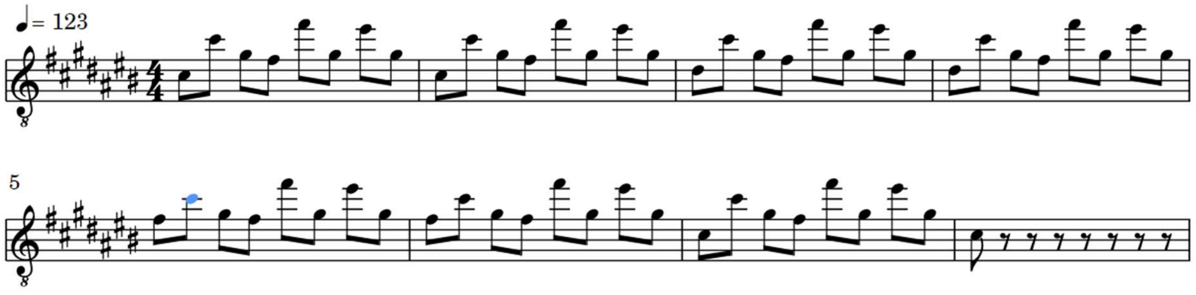

Figure 3: Music Scale for Sweet Child O' Mine

we can see in **figure 3**, this turned out quite well!

4.2 Analysis of Low-Pass and Band-Pass Filters with Comfortably Numb

As you can see in **figure 4,** filtering out the high frequencies and isolating the bass worked well. The low-pass filter was able to cut off all of the frequencies above the mark I had set, and the bass notes are easy to see. There is a little disturbance in between the notes, but that is probably caused by the way the musician is playing instead of our methods. Unfortunately the overtones are not completely filtered out because it was difficult to filter out the overtones without cutting off some of the base notes. With this it is easy to see that the notes played from first to last are B(123 Hz), A(110 Hz), F#(92.5 Hz), and E(82.4 Hz). Even though the frequencies of the notes are very close together, the Fourier Transform and Gabor Transform

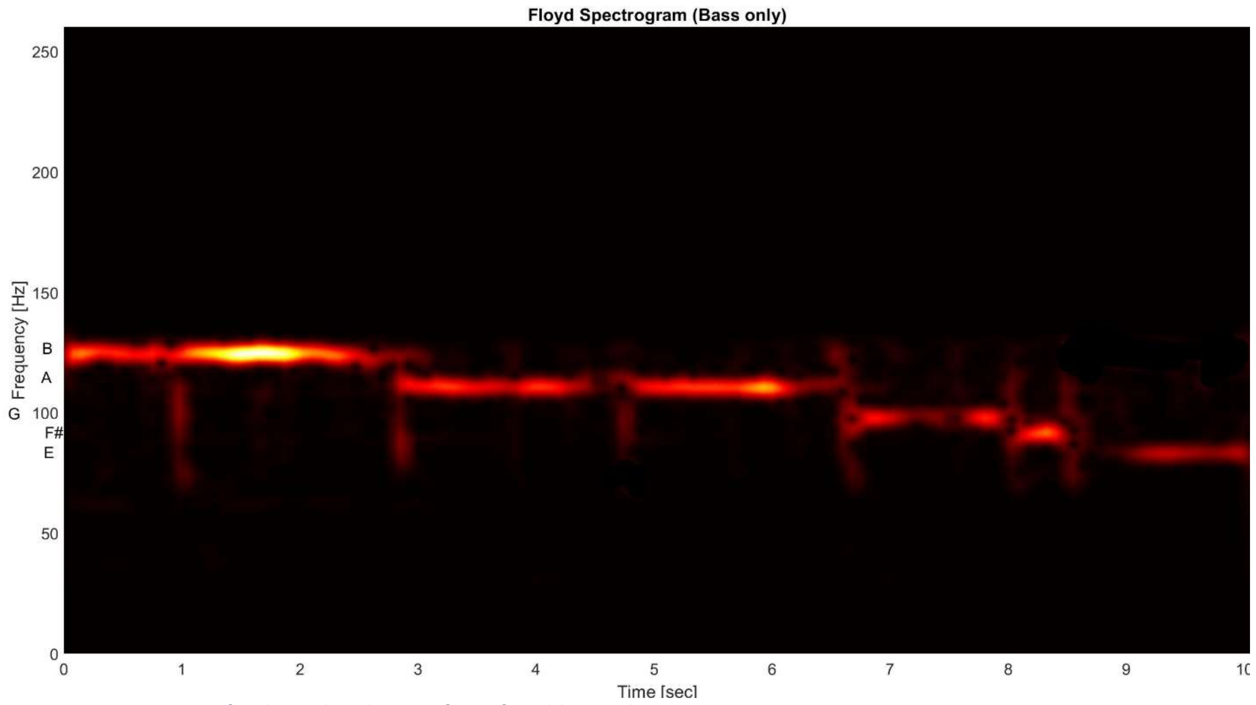

Figure 4: Spectrogram for the Isolated Base of Comfortably Numb

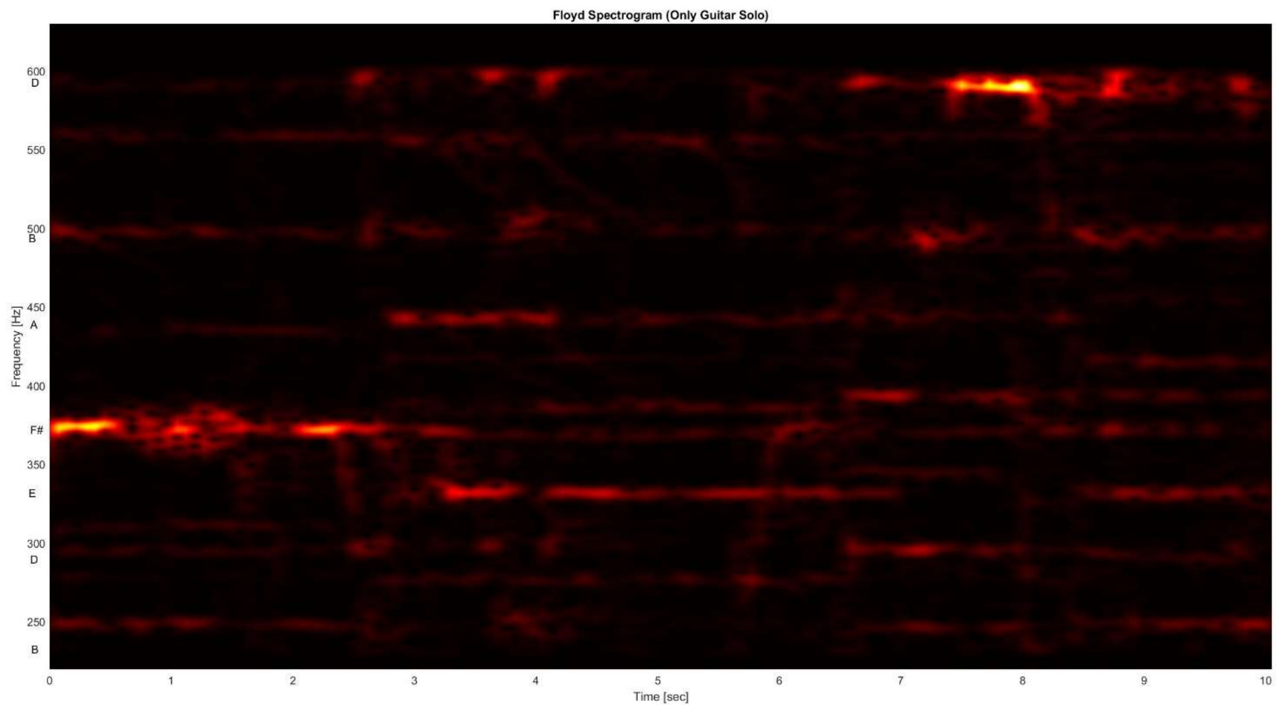

Figure 5: Spectrogram of guitar solo isolated for Comfortably Numb

were able to clearly filter out the noise created from other instruments and the overtones.

Filtering out the guitar solo was more difficult than filtering out the bass because as you can see in **Figure 5,** the overtones from the bass commonly overlap over the notes from the guitar. With this, after using a band-pass filter to remove the overtones and the bass notes as well as possible, the guitar solo can be easily seen. Going from lowest frequency to highest frequency the notes are: B(247 Hz), D(277 Hz), E(330 Hz), F#(370 Hz), A high(440 Hz), B high(494 Hz), and D high(587 Hz). Although the guitar solo is not perfectly filtered, the notes that are being played is obvious to the eye. The only spot of confusion is right at the beginning around the F#. However, this makes sense when listening to the song, because the notes are played rapidly and are not going to equal an exact note (unlike a piano).

5 Summary and Conclusions

The Fourier Transform is a powerful tool for analyzing the frequency content of a signal, but because it maps from the spatial domain to the frequency domain, all spatial information is lost. This is problematic because when the signal is time varying, the relative order of frequencies is important. As a compromise to this, the Gabor Transform is a tool that can be used to limit the loss of spatial resolution, but this comes at a cost of frequency resolution. There is no way to avoid this trade-off between frequency and spatial information, so it is crucial to find the correct window width and sampling rate. This paper highlights the different roles that each of these window parameters have and when to use them. Using these ideas we were able to successfully reconstruct the music score for Gun's and Roses *Sweet Child O' Mine* from a time varying signal (a guitar).

We were also able to create low-pass and band-pass filters to filter isolate certain frequencies of music. A rectangular filter was successfully used to isolate both the bass and the guitar solo in the song *Comfortably Numb* by Pink Floyd. This rectangular filter acted as a low-pass filter to isolate the bass and remove the overtones from the bass. However, even though the rectangular filter was successful at removing the bass and very high overtones from the guitar solo, isolating the guitar solo showed some flaws in the Fourier and Gabor Transform. While sometimes it is obvious what overtones to remove by looking at the spectrogram, it is not always obvious in the overtones that overlap with similar frequencies we are trying to keep. Without any prior knowledge of the song, it can be difficult to know which frequency is an overtone, and which is a note that is actually being played.

6 Appendix A

Below is the following MATLAB functions I used and a brief description of their functions:

abs(x): Returns the absolute value of every element of 'x' or complex magnitude if its complex. I used this function to normalize the data.

Fft(x): Performs a multidimensional Fast Fourier Transform on x, an N-D array. I used this on the data to transform it into the frequency domain.

Fftshift(x): Rearranges the contents by placing the zero-frequency component in the center of the array. If x is a matrix, shifts the first and third quadrant and the second and fourth quadrants. I used this to change our transformed data for proper visualization.

Ifft(x): Performs the multidimensional inverse of the Fast Fourier Transform. I used this after multiplying the centralized filter on the Fourier-transformed data to change it back into the spatial domain.

Ifftshift(x): Inverse of the fftshift function. Used this on the filter to undo the fftshift.

Linspace(x1,x2,n): Creates a vector of n evenly spaced points between x1 to x2.

Max(A): Returns the maximum value of the vector A. Used this for normalizing the data.

pcolor: Takes two vectors which define a grid, setting the x and y tick marks, and a 2D matrix. At each point in the grid, it colors the figure based on the corresponding value in the 2D matrix. We used it to make a spectrogram.

7 APPENDIX B

MATLAB Code

```matlab
clear, clc;
%% GNR code loading data
[y, Fs] = audioread('GNR.m4a');
S = y' ;
n=length(y); L=n/Fs;
t=(1:n)/Fs;
k=(2*pi)/(L)*[0:(n/2-1) -(n)/2:-1];
ks=fftshift(k);
%% Floyd code loading data
[y,Fs] = audioread('Floyd_trim.m4a');
%[y,Fs]=audioread('Floyd.m4a');
%y = y(1:end-1);
n=length(y); L = n/Fs;
t=(1:n)/Fs;
%rescale [-pi pi] domain fft assumes to fit our domain [-L L]
k=(2*pi)/(L)*[0:(n/2-1) -(n)/2:-1];
ks=fftshift(k);
S = y';
St = fft(S);
%% Filtering out high frequency noise
rect= @(x,a)  ones(1,numel(x)).*(abs(x)<a) ; %Creates a rectangular filter around a
%gaus = @(x,a,b) exp(-b*(x-a).^2);
filter = rect(ks,130*2*pi) ; %use 130 Hz as the highest frequency allowed
%filter = gaus(ks,200,0.01);
Sf = St.*fftshift(filter); % apply filter
Sfil = ifft(Sf);

%% Filtering for Guitar solo
rect= @(x,a,b)  ones(1,numel(x)).*( abs(x) < b & abs(x) > a) ; % creates a band pass rectangle filter
filter = rect(ks, 230*2*pi, 600*2*pi ); %frequencies between 230 and 600
Sf = St.*fftshift(filter);
Sfil = ifft(Sf); % apply filter
plot(t,filter) %plot filter
%% Plot waveform and fft
subplot(2, 1, 1);
plot(t, S);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Waveform Signal');

subplot(2, 1, 2);
plot(ks, abs(fftshift(St))/max(abs(St)), 'r');
```

```
set(gca, 'XLim', [0 2e3]);
xlabel('Frequency [Hz]');
ylabel('Amplitude');
title('Fast Fourier Transform');

sgtitle('GNR Waveform and FFT');

saveas(gcf, 'GNR_Waveform_FFT.jpg');
close(gcf);
%% Testing for different window sizes
% Plot spectrogram over varying window sizes
a = [100; 150; 250; 350; 500; 600; 750; 1000];
tslide = 0:0.1:L; %creates vector to "slide" across
%initialize matrix for spectrogram and max frequencies
Sgt_spec = zeros(length(tslide), n);
max_k = zeros(1, length(tslide));

for j=1:length(tslide)
    g = exp(-a .* (t - tslide(j)) .^ 2); %gabor filter with gaussian
    %g=exp(-a*(t - tslide(j)).^10); %super gaussian filter
    %g=(1-(t - tslide(j)).^2).*exp(-a*(t - tslide(j)).^2 ./ 2);
%mexican
    %hat filter
    Sg = g .* S; %Use Sfil for Floyd Songs
    for k=1:length(a)
        Sgt = fft(Sg(k, :));
        Sgt_spec(j, :, k) = fftshift(abs(Sgt));
    end
end

for j=1:length(a)
    %Plot Spectrogram
    f = figure(j);
    pcolor(tslide, ks/(2*pi), Sgt_spec(:, :, j).');
    shading interp;
    set(gca,'Ylim',[0 500]);
    colormap(hot);
    xlabel('Time [sec]');
    ylabel('Frequency [Hz]');
    title(sprintf('GNR Spectrogram with Window Size %d', a(j)));
    set(gca , 'fontsize' ,25);
    saveas(f, sprintf('GNR_a=%d.jpg', a(j)));
    close(f);
end
%% Testing for different Time Increments
a = 150;
for dt=[0.05, 0.075, 0.1, 0.2, 0.5]

    tslide = 0:dt:L;
    Sgt_spec = zeros(length(tslide), n);
    notefreq = zeros(1,length(tslide));
    for j=1:length(tslide)
        g = exp(-a * (t - tslide(j)).^ 2);
        %g=exp(-a*(t - tslide(j)).^10);
        %g= abs(t - tslide(j)) <= 1/(2*a);
        %g = (abs(t-tslide(j)) < 1);
        Sg = g .* S; %Use Sfil for Floyd songs
        Sgt = fft(Sg);
        [Max,Ind] = max(fftshift(abs(Sgt)));
        Sgt_spec(j, :) = fftshift(abs(Sgt));
        notefreq(j) = abs(ks(Ind)/(2*pi));
    end
    f=figure(1)
    pcolor(tslide, ks/(2*pi), Sgt_spec(:, :).');
    shading interp;
    %set(gca,'Ylim', [0 260]);
    set(gca,'Ylim',[220 630]);
    colormap(hot);
    xlabel('Time [sec]');
    ylabel('Frequency [Hz]');
    title(sprintf('Floyd Spectrogram (Only Bass)'));
    %gtext({'C5#';'D#';'F#';'G#';'#C';'F';'#F'})
    saveas(f, sprintf('Floyd_dt=%f.jpg', dt));
    close(f);
end

%% Code for GNR
%same code as above, but tests only one dt and window
a = 150;
for dt=0.1

    tslide = 0:dt:L;
    Sgt_spec = zeros(length(tslide), n);
    notefreq = zeros(1,length(tslide));
    for j=1:length(tslide)
        g = exp(-a * (t - tslide(j)).^ 2);
        Sg = g .*S;
        Sgt = fft(Sg);
        [Max,Ind] = max(fftshift(abs(Sgt)));
        Sgt_spec(j, :) = fftshift(abs(Sgt));
        notefreq(j) = abs(ks(Ind)/(2*pi));
    end
    f=figure(1)
    %create spectrogram for GNR
    pcolor(tslide, ks/(pi), Sgt_spec(:, :).');
    shading interp;
    %Create axis for y for better readability
    set(gca,'Ylim',[500 1500]);
    set(gca , 'fontsize' ,15);
    colormap(hot);
    xlabel('Time [sec]');
    ylabel('Frequency [Hz]');
    title(sprintf('GNR Spectrogram with Notes'));
    %Showcase where what each hotspot frequency is
    gtext({'C5#';'D#';'F#';'G#';'#C';'F';'#F'})
    saveas(f, sprintf('GNR Spectrogram with Notes.jpg'), dt);
    close(f);
end
%% Code for Floyd Song
%same code as above, but with the filter applied to Floyd
for dt=0.075

    tslide = 0:dt:L;
    Sgt_spec = zeros(length(tslide), n);
    notefreq = zeros(1,length(tslide));
    for j=1:length(tslide)
        g = exp(-a * (t - tslide(j)).^ 2);
        %apply filterered frequency to gabor
        Sg = g .* Sfil;
        Sgt = fft(Sg);
        [Max,Ind] = max(fftshift(abs(Sgt)));
        Sgt_spec(j, :) = fftshift(abs(Sgt));
        notefreq(j) = abs(ks(Ind)/(2*pi));
    end
    f=figure(1)
    pcolor(tslide, ks/(2*pi), Sgt_spec(:, :).');
    shading interp;
    %set(gca,'Ylim', [0 260]); %Use this range if we are doing the bass
    set(gca,'Ylim',[220 630]); %Use this range for the guitar solo
    set(gca ,'fontsize' ,15);
    colormap(hot);
    xlabel('Time [sec]');
    ylabel('Frequency [Hz]');
    %title(sprintf('Floyd Spectrogram (Only Guitar Solo)'));
    title(sprintf('Floyd Spectrogram (Bass only)'));
    gtext({'B';'D';'E';'F#';'A';'B';'D'},'fontsize',15)
    saveas(f, sprintf('Floyd Spectrogram Guitar Solo.jpg'));
    close(f);
end
%% Make all Notes
%makes a vector of all the notes
max_freqs = notefreq ;
notes = ["A", "A#", "B", "C", "C#","D", "D#", "E", "F", "F#", "G", "G#"];
%designates frequencies to each of these notes
fund_freqs = [27.5, 29.135, 30.863, 32.703, 34.648,36.708, 38.891, 41.203, 43.654,46.249, 48.99, 51.913];

allnotes = [];
allfreqs = [];

for j = 1:8
    for note = notes
        allnotes = [allnotes  strcat(note, num2str(j))];
    end
    for freq = fund_freqs
        %find frequencies at each octave
        % freqs double each time you move up scale
        allfreqs = [allfreqs freq*(2^(j-1))];
    end
end
%%
%create a final music score for the songs
music_score = [];

for freq = max_freqs
    %matches the frequencies we got to playable notes
    [min_val, index] = min(abs(freq - allfreqs));
    music_score = [music_score allnotes(index)];
end

```