AMATH 482 - Assignment 2

Clarisa Leu-Rodriguez

Winter 2021 - February 10th, 2021

Abstract

In this paper, we explore a portion of two of the greatest rock and roll songs of all time, Sweet Child O' Mine by the Guns N' Roses and Comfortably Numb by Pink Floyd, in order to analyze the music signals and identify the notes being played in each song from different instruments. Our aim in this exploration is to demonstrate the usefulness of the Gabor Transform in extracting frequency information from time-varying signals, while preserving information about where those frequencies occur in time. We explore the ideas of oversampling and under-sampling signals, the Fourier Transform, the Gabor Transform, and window functions in this paper - as well as provide directions for future work on this problem.

Section I. Introduction and Overview

We are given two portions from two of the greatest rock and roll songs of all time - *Sweet Child O' Mine* by the Guns N' Roses (≈ 14 seconds long) and *Comfortably Numb* by Pink Floyd (≈ 60 seconds long). Our goal when analyzing these pieces of music is, through the use of the Gabor transform with a Gaussian window function & varying its width/time-step, to reproduce the music score for the guitar in the GNR music piece, the bass in the Pink Floyd music piece, and portions of the guitar in the Pink Floyd music piece.

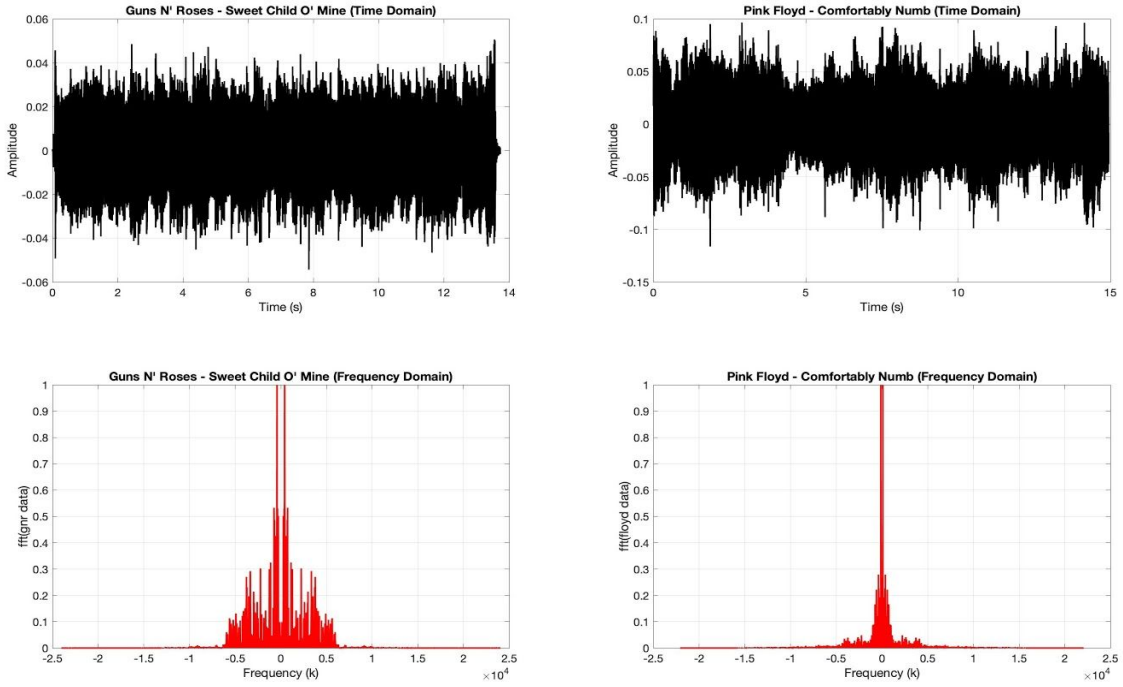

Figure 1: Original, unfiltered song clips in the time and frequency domain. The left-most plots (Figure 1a on the top half and Figure 1c on the bottom half) show the time and frequency data for the GNR piece and the right-most plots (Figure 1b on the top half and Figure 1d on the bottom half) show the time and frequency data for the Floyd piece (note that here we are only showing the first ≈ 15 seconds of the Floyd clip).

Section II. Theoretical Background

In order to successfully isolate the music score of the bass in the GNR music piece, the bass in the Pink Floyd music piece, and portions of the guitar music score in the Pink Floyd piece, we use the Gabor transform (which is a special case of the short-time Fourier transform) and Gaussian filtering. In this section, I will discuss the mathematics and theory behind our solution.

Section II.I. The Fourier & Gabor Transform

Recall that the Fourier transform is the primary tool for converting between the spatial and frequency domains of a signal. Where, given a periodic function $f(x)$ with $x \in R$, the Fourier transform of $f(x)$, written $\widehat{f}(k)$, is defined by the formula:

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

Equation 1: The Fourier transform of $f(x)$.

Additionally, we use the inverse Fourier transform to transform from the frequency domain back to the spatial domain, where given $\widehat{f}(k)$ in the frequency domain, we can recover $f(x)$:

$$f(x) = \frac{1}{\sqrt{2 \cdot pi}} \int_{-\infty}^{\infty} \widehat{f}(k) \cdot e^{ikx} dk$$

Equation 2: The inverse Fourier transform of $\widehat{f}(k)$.

One shortfall of the Fourier transform in time-frequency analysis is that it only gives us a global measurement of the frequency content of a signal - meaning that valuable information in signals which vary in time (e.g. a song) is lost when we do a Fourier transform. That is, when we do a Fourier transform - we get complete frequency resolution but lose all spatial information. This can be clearly seen in Figure 1 when performing a pure fast Fourier transform on the music clips. The Gabor transform seeks to recover information in the time domain which is lost when doing a pure Fourier transform and allows us to understand how the frequency of a signal varies over time.

Formally, given some signal $f(t)$ in the time domain, the Gabor transform, written as $G_f(\tau,\ k)$, is defined as the following function:

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

Equation 3: The Gabor transform of $f(t)$.

In Equation 3, $g(t)$ is defined to be the filter function and acts as a sliding window in the time domain - amplifying the signals within the window near its center τ, and diminishing the frequencies outside of the window. A very common filter function used in the Gabor transform is the Gaussian function - due to its smoothness and how values outside of its center quickly decay to zero. Another window function is the Shannon filter (step-function), which sets frequencies outside of a defined range to zero. In theory, we can use any function as our window function so long as it satisfies our assumptions for $g(t)$ - where two common assumptions made are that $g(t)$ is real and symmetric, and the L_2-norm of $g(t)$ is set to the total energy of a function. Our choice of window function to explore for this paper is the Gaussian function, where the equation for our Gaussian window function is:

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

Equation 4: The equation for the general Gaussian window filter function, centered at τ with a width of a.

The inverse of the Gabor transform is also given by the following equation, where given $G_f(\tau, k)$, we can recover $f(t)$:

$$f(t) = \frac{1}{2 \cdot pi \cdot ||g||_2} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} G_f(\tau, k) g(t-\tau) e^{ikt} dk d\tau$$

Equation 5: The inverse Gabor transform.

When picking an appropriate window function when performing a Gabor transform, it is also important to note that each window function will decay at a different rate and have their own specific properties. We picked a Gaussian window function for this problem due to its smooth nature - although future work for this paper would be to explore different window functions. In general, you should pick a window function which best fits the signal and task you wish to accomplish.

Section II.II. Time-Frequency Analysis - Finding a Balance

In time-frequency analysis, the goal is to strike a balance between understanding the information about how a signal varies in both time and frequency. There is a tradeoff between the two - as we cannot have all the information about both time and frequency, and the more information we extract from a signal about its frequency signatures - the less we know about how those vary in time. This is the Heisenberg Uncertainty Principle in action. The Fourier transform is at one end of the extreme - giving us all the information about frequency of a signal but losing all of the information about time. Using the Gabor transform, as stated previously, we can preserve information about how a signal's frequency varies over time by picking an appropriate window function and an appropriate width and time-step for our window function through an exploratory process.

By varying the width of our window function in our Gabor transform, i.e. a in Equation 4, and varying the window width, we'll call this $\Delta\tau$, as we slide along our function $f(t)$ in Equation 3 - we aim to extract adequate information on both time and frequency from our signal. When picking an appropriate width, a, for our window function - as the window width increases this means that a larger range of frequencies will be captured in our window, but the position in time of these frequencies might also be lost. When picking an appropriate $\Delta\tau$, as we increase $\Delta\tau$, we risk missing portions of the signal if we have too large of a time-step. When sampling from our data, we don't want to under sample (having too large of a width a and/or too small of a time-step $\Delta\tau$) or over sample (having too small of an a and/or too large of a time-step $\Delta\tau$) as our goal is to balance the resolution of our signal in both the time and frequency domains. Note that as we increase a - our window becomes more narrow and vice versa for when we decrease a.

Section III. Algorithm Implementation and Development

To start this problem, we first import the music pieces into MATLAB and listen to how they sound. In the GNR piece - the guitar is clearly audible in the piece with no other instruments being played. In the Floyd piece - we hear multiple instruments throughout the clip, some audibly recognizable as the bass, the electric guitar, an additional rhythm guitar, the drums, and the keyboard. In addition, as the Floyd piece was quite long, we only analyze a portion of the song, particularly the first 15 seconds of the piece. We also define a vector k for each piece - which defines the range of frequencies. As our sound data is in Hz, we also scale this vector by $1/L$ (where L is the sampling rate of the clip) since MATLAB's `fft` function assumes $2 \cdot pi$ periodic frequencies.

To reproduce the music score of the guitar in the GNR piece, we use the Gabor transform with a Gausssian function as our window function. We explore a range of values for our possible width a - 1, 10, 100, and 1000 and a range of candidate $\Delta\tau$ values for our time-step (i.e. how we slide along our data in the time dimension) - 0.1, 0.2, 0.5, and 1. For each value of a and $\Delta\tau$ - we plot the corresponding spectragram they produce and visually choose one which shows us the most information about how the songs frequencies vary over time. This was through the use of making spectrograms, which show us how a signal's frequency varies over time.

Overall, we determined the best a value was 100 and the best $\Delta\tau$ value was 0.2 from the candidates. Next - to remove the overtones (i.e. - integer multiples of the fundamental frequency) from the frequency data of the GNR piece, we use a Gaussian filter centered around the maximum frequency for each time slice. After doing this for each time slice, we are able to reproduce the music score for the guitar solo in the GNR music piece.

To reconstruct the bass's music score from the Floyd music piece, we also use the Gabor transform with a Gaussian function as our window function and explore the same range of candidate width values a - 1, 10, 100, and 1000 and same range of $\Delta\tau$ values for our time-step - 0.1, 0.2, 0.5, and 1. For each value of a and $\Delta\tau$ - we also plot the corresponding spectragram and exploritorily determine which pair of a, $\Delta\tau$ show us the most information about how the music piece's frequencies vary over time. After determining the best pair of a, $\Delta\tau$ which shows us the most information about how the songs frequencies vary over time ($a = 100$ and $\Delta\tau = 0.1$) - we also manually construct a vector of frequencies which look the brightest/show a pattern and are within the range a bass makes (this range is in general roughly 40-400Hz for a bass guitar). The vector we constructed contains frequencies within the range of 80-130 Hz.

After constructing this vector - as there are multiple instruments being played within the Floyd piece, we try to apply a Gaussian filter in frequency space centered around the bass notes for each time slice centered around the note the bass is/might be playing in the first harmonic. This is to filter out any overtones of the bass - as well as any other noise other instruments might be playing. This portion of this problem was the most difficult as sometimes all of the instruments are playing in the same tune based on the audio sample. However, to do this, we defined a helper function `get_fundamental_freq` which takes in a frequency, the range of frequencies we determined by looking at our spectragram the bass is most likely to be playing, a specified error bound - and we iteratively check if the passed in frequency is within 1, 2, 3, 4, 5, or 6 harmonics of any of the frequencies we determined our bass to most likely be playing within a certain error range - and simply return the first harmonic of that frequency if it is within the specified range. That is, we can determinitely say within a specified error range the note the bass is most likely playing for each time slice and use this note to filter around in frequency space with a Gaussian function to filter out the overtones of the bass - as well as other instruments in the piece. After doing this, we were able to isolate and reproduce the music score for the bass in the Floyd piece.

To put together the pieces of the guitar solo in the Floyd piece, we also use the notes we determined the bass to be playing for each time slice and specifically zero these frequencies and their respective overtones out in our Floyd piece in the frequency domain to start. After filtering out the bass in frequency space - we then use the Gabor transform with a Gaussian window function on our transformed data set (i.e. - Floyd piece with the bass notes and their overtones removed). The process for determining the best pair of a, $\Delta\tau$ is the same for other portions of this section - where we determine $a = 1000$ and $\Delta\tau = 0.1$ to visually look the best at conveying how the frequencies vary over time for higher frequencies (notes the guitar is most likely playing). Since we have filtered out the bass notes, when applying a Gaussian filter in frequency space - we then center around the maximum frequency for each time slice and do not worry if it is a bass note. Through these methods, we were able to construct a portion of the music score for the guitar - although more work can be done to reconstruct a larger portion of the guitar's music score with more accuracy.

Section IV. Computational Results

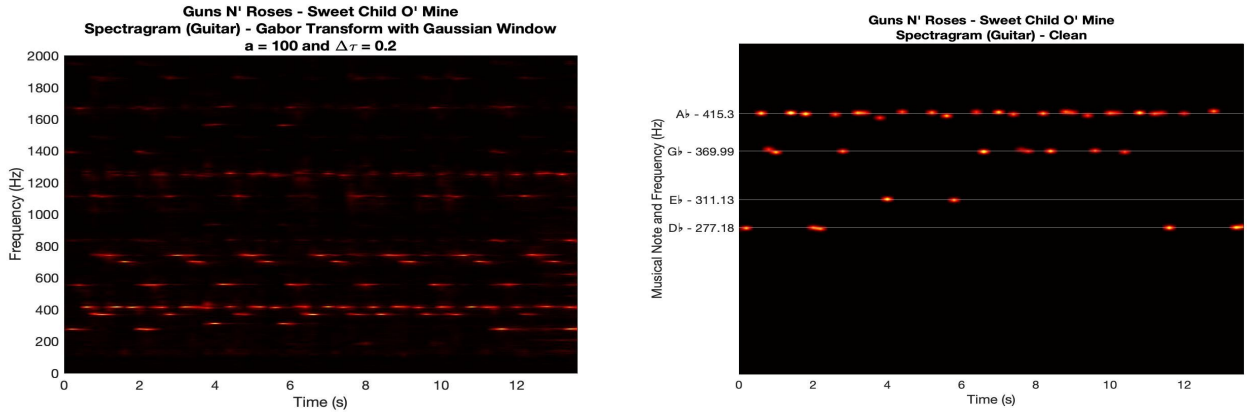

Figure 2: The image on the left (Figure 2a) shows the spectrogram of the GNR music piece after the Gabor transform with the specified with and time-step. The image on the right (Figure 2b) shows the spectrogram of the GNR music piece after using a Gaussian filter to filter out the overtones of the guitar - along with the respective notes labeled on the y-axis and their corresponding frequencies.

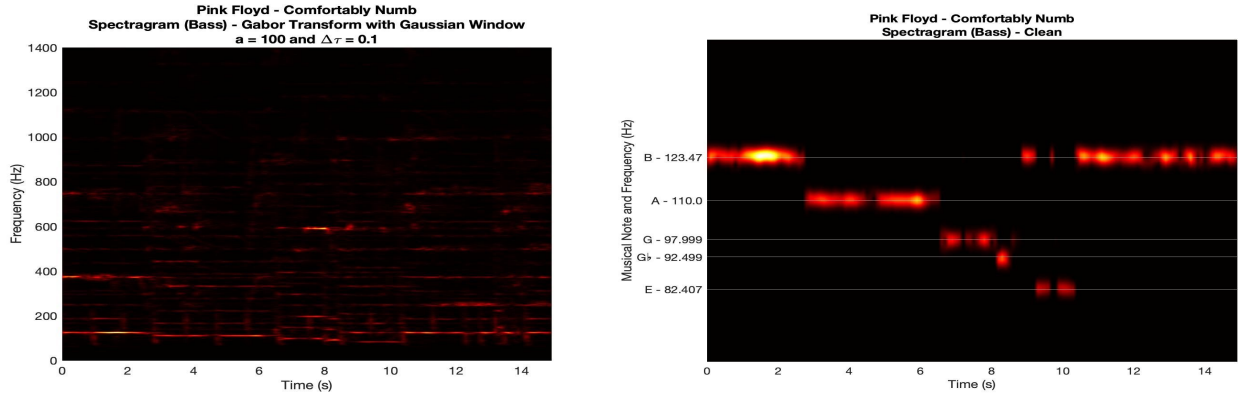

Figure 3: The image on the left (Figure 3a) shows the spectrogram of the Floyd music piece after the Gabor transform with the specified with and time-step. The image on the right (Figure 3b) shows the spectrogram of the Floyd music piece after using a Gaussian filter to filter out the overtones of the bass, centered along the notes we suspected the bass to be playing.

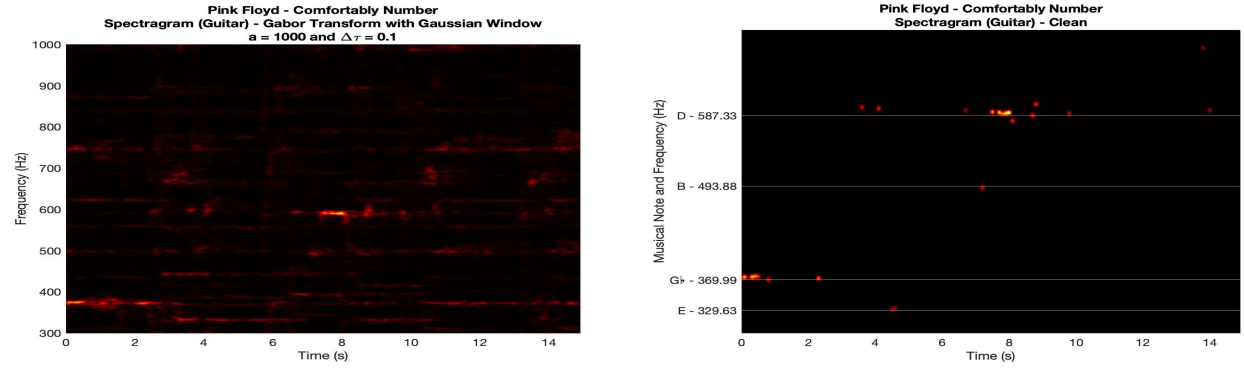

Figure 4: The image on the left (Figure 4a) shows the spectrogram of the Floyd music piece after the Gabor transform with the specified with and time-step (hiding the bass line). The image on the right (Figure 4b) shows the spectrogram of the Floyd music piece after using a Gaussian filter to filter out the overtones of the guitar - along with the respective notes we suspect the guitar to be played labeled.

Section V. Summary and Conclusions

Through the use of the Fourier transform, the Gabor transform (with a Gaussian window function), and Gaussian filtering in the frequency domain - we determine the music score for the guitar in *Sweet Child O' Mine* by the Guns N' Roses (shown in Figure 2b), a portion of the music score for the bass in *Comfortably Numb* by Pink Floyd (shown in Figure 3b) and a portion of the music score for the guitar in *Comfortably Numb* by Pink Floyd (shown in Figure 4b).

Analyzing the music score for the guitar in the GNR piece - we see that this is the cleanest music score we were able to produce of the three. The music score for the guitar in the GNR piece was the easiest of the three to reproduce as well - mainly due to the fact that the clip only features the guitar being played. Compared to the Floyd piece - as stated previously - the Floyd piece was complicated in a few ways, the main one being that there were multiple instruments being played throughout the piece and also that they would sometimes all play in tune - making it hard to isolate one instrument's frequency from another.

Analyzing the music score for the bass in the Floyd piece - our results are also pretty clean. This was the second easiest music score to reproduce of the three - as the bass line was very apparent throughout the piece, which can be seen in the spectrogram in Figure 3a (the bright line within the range of 80-130 Hz). We also note that the bass throughout the original piece given is not changing very much throughout the 60 seconds - and the bass line always has this "staircase" pattern throughout. This was confirmed by plotting different 15 second sections of the given 60 second clip - and plots can be reproduced very easily (see Appendix B: MATLAB Codes) for those interested.

Analyzing the music score for the guitar in the Floyd piece - we see that our methods of filtering out the overtones of the bass could be improved, as we see the music score contains some of the overtones of the bass - particularly G flat, E, and B. However, as the guitar and the bass play in synchrony during portions of this song (confirmed by listening to the music clip) - the guitar could also be likely playing these notes as well - as this is certainly within the audible frequency range of a guitar.

Future work for this problem would be to explore the two music pieces with different window functions. A Shannon function would be a good place to start, as we can simply filter around frequencies we know a particular instrument plays. It would also be interesting to reconstruct the entirety of the bass line and guitar solo for the Floyd piece, however due to computational constraints only the first 15 seconds are shown in this paper. In addition - more work could be done on improving the methods used to filter out the overtones of the bass in the Floyd piece when trying to isolate the guitar and improving the error bound on the notes we suspect the bass to be playing over time.

Section VI. References

- https://www.groundguitar.com/slash-gear/ - I used this source to understand the type of guitarist Slash played in *Sweet Child O' Mine* as electric guitars and acoustic guitars have different frequency ranges in order to check my work.
- https://www.studybass.com/gear/bass-tone-and-eq/bass-frequency-range/ - I used this source to find the frequency range for a bass.
- In addition, the course notes provided on the course website and all lecture material was used. I also used the given music score on the assignment specification to match the frequencies to the musical note on my spectrograms.

Appendix A. MATLAB Functions Used and Brief Implementation Explanation

This appendix contains a list of the MATLAB functions used and additional information regarding how it was used to solve the given problem. Additional information can also be found in Section II. Theoretical Background and also Section III. Algorithm Implementation and Development. Note, descriptions of functions were sourced from the MATLAB documentation which can be found here online: https://www.mathworks.com/help/matlab/.

- `[Y, Fs] = audioread(FILENAME)` reads an audio file specified by FILENAME and returns the sampled data in Y and the sample rate FS, in Hertz. We used this to import the sound clips into MATLAB.
- `floor(X)` rounds X to the nearest integer towards minus infinity. When breaking up the sampled data from our Floyd piece into smaller portions (in our case - we break it up into four portions and look at the first portion of the four which represents the first 15 seconds of the clip) - we used this to get an integer value to use as an index into our the sampled data returned from `audioread`.
- `fftshift(X)` shifts the zero-frequency component to the center of the spectrum. If X is a vector, it swaps the left and right halves of X. For matrices, the function `fftshift` swaps the first and third quadrants and the second and fourth quadrants. We used `fftshift` to better visualize the Fourier transform with the zero-frequency component in the middle of the spectrum.
- `zeros(M, N, P, ...)` generates an M-by-N-by-P-by-... array of zeros. We use the `zeros` function in MATLAB to preallocate memory before computation - which saves the computer time as it doesn't need to grab more buckets during computation as the space to store the result has already been preallocated.
- `[M, I] = max(X)` returns the largest element in the vector X (M) and also the index I corresponding to its position in X. We used this to find the largest value in our data for each time slice in order to use a Gaussian filter centered around this value in the frequency domain.
- `abs(X)` is the absolute value of the elements of X. When X is complex, it computes the complex modulus (magnitude) of the elements of X. We used this in our code when we wanted to find the maximum value in magnitude within our data for each time slice - as well as for plotting/analyzing only positive frequencies in our data.
- `fft(X)` returns the discrete Fourier transform (DFT) of a vector X. We used this to convert our time data into the frequency domain.
- `ifft(F)` return the inverse discrete Fourier transform of a vector X. We used this to convert our frequency data back into the time domain when filtering out frequencies of the bass in Part #3 of the MATLAB Codes.
- `ifftshift(X)` swaps the left and right halves of X and undoes the effects of `fftshift`. We used this to undo the `ifft` in Part #3 of the MATLAB Codes when filtering out the bass from the Floyd piece.
- `pcolor(X, Y, C)` makes a pseudocolor plot on the grid defined by X and Y - where X and Y are vectors. We used this for plotting spectrograms.
- `audioplayer(Y, Fs)` creates an audioplayer object for signal Y, using a sample rate Fs. A handle to the object is returned. We used this to play our sound data in combination with using the MATLAB function `playblocking` on the returned handle.
- `yticks(ticks)` specifies the values for the tick marks along the y-axis of the current axes - where ticks is specified in increasing order. We used this to label the musical notes on our spectrogram - in combination with `yticklabels` and `yline`.

Appendix B. MATLAB Codes

This appendix contains the MATLAB code used in this paper. Please refer to Appendix A. MATLAB Functions Used and Brief Implementation Explanation for more information regarding how each of the MATLAB functions used in our code below works.

assignment_2.m

```
%{
    Assignment #2 - Rock & Roll and the Gabor Transform
    AMATH482 - Computational Methods For Data Science - Feb. 10th, 2021
    Taught by Professor Jason J. Bramburger (Winter 2021)
    Written By: Clarisa Leu-Rodriguez - email: cleu@uw.edu

    This MATLAB script is written with the intentions of being ran in
    sections, in order to explore the Gabor transform with a Gaussian
    window function.
    Please read each respective section for instructions on how to use/modify.
%}

%% Clear and refresh the workspace.
clear; clc; close all;


%% Import Data
[y_gnr, Fs_gnr] = audioread('GNR.m4a');
y_gnr = y_gnr';
tr_gnr = length(y_gnr) / Fs_gnr; % record time in seconds.
t_gnr = (1:length(y_gnr)) / Fs_gnr;

[y_floyd, Fs_floyd] = audioread('Floyd.m4a');
y_floyd = y_floyd';
% Uncomment to analyze whole piece & get rid of the last data point so it
% is of even length for plotting.
% y_floyd = y_floyd(1:end-1);

% We look at a smaller sample of the Pink Floyd song as it is
% computationally expensive to analyze the whole clip at once.
len_floyd = floor(length(y_floyd) / 4);
% Modify the length of the floyd clip here.
% e.g. y_floyd((2*len_floyd + 1) : (3*len_floyd)) analyzes 30-45 seconds.
y_floyd = y_floyd(1 : len_floyd);
tr_floyd = length(y_floyd) / Fs_floyd; % record time in seconds.
t_floyd = (1:length(y_floyd)) / Fs_floyd;


%% Plot unfiltered music signals in time/frequency space
close all;
figure();
```

```
% Plot time.
subplot(2, 2, 1);
plot((1:length(y_gnr)) / Fs_gnr, y_gnr, 'k', 'Linewidth', 2);
xlabel('Time (s)'); ylabel('Amplitude');
title('Guns N'' Roses - Sweet Child O'' Mine (Time Domain)');
grid on;
set(gca, 'Fontsize', 14);

subplot(2, 2, 2);
plot((1:length(y_floyd)) / Fs_floyd, y_floyd, 'k', 'Linewidth', 2);
xlabel('Time (s)'); ylabel('Amplitude');
title('Pink Floyd - Comfortably Numb (Time Domain)');
grid on;
set(gca, 'Fontsize', 14);

% Plot Frequency.
n_gnr = length(y_gnr);
L_gnr = tr_gnr;
% Scale k vector by 1/L because of Hz.
k_gnr = (1/L_gnr).*[0:n_gnr/2-1 -n_gnr/2:-1];
ks_gnr = fftshift(k_gnr);
yt_gnr = fft(y_gnr);

subplot(2, 2, 3);
plot(ks_gnr, abs(fftshift(yt_gnr))/max(abs(yt_gnr)), 'r', 'Linewidth', 2);
xlabel('Frequency (k)'); ylabel('fft(gnr data)');
title('Guns N'' Roses - Sweet Child O'' Mine (Frequency Domain)');
grid on;
set(gca, 'Fontsize', 14);

n_floyd = length(y_floyd);
L_floyd = tr_floyd;
 % Scale k vector by 1/L because of Hz.
k_floyd = (1/L_floyd).*[0:n_floyd/2-1 -n_floyd/2:-1];
ks_floyd = fftshift(k_floyd);
yt_floyd = fft(y_floyd);

subplot(2, 2, 4);
plot(ks_floyd, abs(fftshift(yt_floyd))/max(abs(yt_floyd)), 'r', 'Linewidth',
2);
xlabel('Frequency (k)'); ylabel('fft(floyd data)');
title('Pink Floyd - Comfortably Numb (Frequency Domain)');
grid on;
set(gca, 'Fontsize', 14);


%% Play song clip - GNR
 p8_gnr = audioplayer(y_gnr, Fs_gnr); playblocking(p8_gnr);
```

```
%% Play song clip - Pink Floyd
 p8_floyd = audioplayer(y_floyd, Fs_floyd); playblocking(p8_floyd);


%% Part #1 - Explore GNR - Gaussian Window Function - Find Good Width.
% This section can also be modified to find a good width for the Gaussian
% window function using the Pink Floyd clip.

close all;
time_step = 0.2;  % Modify time step here.
tau_gnr = 0:time_step:L_gnr;
a_gnr_vals = [1 10 100 1000];  % Explore different a values
                               % for the Gaussian window.
figure();
for i = 1:length(a_gnr_vals)
    a_gnr = a_gnr_vals(i);
    ygt_spec_gnr_gabor = zeros(length(tau_gnr), n_gnr);
    for j = 1:length(tau_gnr)
        % Using a Gaussian Window Filter.
        g = exp(-a_gnr*(t_gnr-tau_gnr(j)).^2);
        yg = g.*y_gnr;
        ygt = fft(yg);
        ygt_spec_gnr_gabor(j, :) = abs(fftshift(ygt));
    end
    % Plot spectrogram.
    subplot(2, 2, i);
    pcolor(tau_gnr, ks_gnr, ygt_spec_gnr_gabor.');
    shading interp;
    title({['GNR - Sweet Child O'' Mine' ]
        ['Gabor Transform with Gaussian Window']
        ['a = ' num2str(a_gnr) ' and \Delta\tau = ' num2str(time_step)]
        });
    colormap(hot);
    ylabel('Frequency (Hz)');  xlabel('Time (s)');
    set(gca,'ylim',[0, 2000],'Fontsize', 14);
end


%% Part #1 - Explore GNR - Gaussian Window Function - Find Good Time Step.
% This section can also be modified to find a good time step for the Gaussian
% window function using the Pink Floyd clip.

close all;
% Parameters for Gaussian Window Filter Function.
time_step_vals = [0.1 0.2 0.5 1];  % Explore different time steps.
a_gnr = 100;  % Modify width here.
figure();
for i = 1:length(time_step_vals)
```

```
    time_step = time_step_vals(i);
    tau_gnr = 0:time_step:L_gnr;
    ygt_spec_gnr_gabor = zeros(length(tau_gnr), n_gnr);
    for j = 1:length(tau_gnr)
        % Using a Gaussian Window Filter.
        g = exp(-a_gnr*(t_gnr-tau_gnr(j)).^2);
        yg = g.*y_gnr;
        ygt = fft(yg);
        ygt_spec_gnr_gabor(j, :) = abs(fftshift(ygt));
    end
    % Plot spectrogram.
    subplot(2, 2, i);
    pcolor(tau_gnr, ks_gnr, ygt_spec_gnr_gabor.');
    shading interp;
    title({['GNR - Sweet Child O'' Mine' ]
        ['Gabor Transform with Gaussian Window']
        ['a = ' num2str(a_gnr) ' and \Delta\tau = ' num2str(time_step)]
        });
    colormap(hot);
    ylabel('Frequency (Hz)');  xlabel('Time (s)');
    set(gca,'ylim',[0, 2000],'Fontsize', 14);
end


%% Part #1 - Explore GNR - Gaussian Window Function Generate Final Plot.
close all;
time_step = 0.2;  % Modify time step here.
tau_gnr = 0:time_step:L_gnr;
a_gnr = 100;  % Modify width here.
ygt_spec_gnr = zeros(length(tau_gnr), n_gnr);
ygt_spec_gnr_gabor = zeros(length(tau_gnr), n_gnr);
gnr_guitar_notes = zeros(1, length(tau_gnr));

for j = 1:length(tau_gnr)
    % Using a Gaussian Window Filter.
    g = exp(-a_gnr*(t_gnr-tau_gnr(j)).^2);
    yg = g.*y_gnr;
    ygt = fft(yg);
    ygt_spec_gnr_gabor(j, :) = abs(fftshift(ygt));

    % Find maximum frequency for each time slice - this is the note played.
    [~, i] = max(abs(ygt));
    gnr_guitar_notes(j) = k_gnr(i);

    % We then can use a Gaussian filter to filter around the maximum frequency
    % in the frequency domain for a cleaner looking spectragram if we want.
    gaussian_tau = 0.1;  % We pick a tau here which looks best.
    gaussian_filter = exp(-gaussian_tau*(k_gnr-gnr_guitar_notes(j)).^2);
    ygt_filtered = gaussian_filter.*ygt;
```

```
    ygt_spec_gnr(j, :) = abs(fftshift(ygt_filtered));
end

% Plot clean spectrograms (Gabor here).
figure();
pcolor(tau_gnr, ks_gnr, ygt_spec_gnr_gabor.');
shading interp;
title({['Guns N'' Roses - Sweet Child O'' Mine' ]
    ['Spectragram (Guitar) - Gabor Transform with Gaussian Window']
    ['a = ' num2str(a_gnr) ' and \Delta\tau = ' num2str(time_step)]
    });
colormap(hot);
ylabel('Frequency (Hz)');  xlabel('Time (s)');
set(gca,'ylim',[0, 2000],'Fontsize', 14);

% Plot clean spectrograms (filtered, finished version - Gabor and Gaussian
% filter in freq. space).
figure();
pcolor(tau_gnr, ks_gnr, ygt_spec_gnr.');
shading interp;
title({['Guns N'' Roses - Sweet Child O'' Mine' ]
    ['Spectragram (Guitar) - Clean']
    });
colormap(hot);
% Show notes on y-axis (pulled off visually by looking at spectragram).
yticks([277.18, 311.13, 369.99, 415.30])
yline(277.18, 'w'); yline(311.13, 'w'); yline(369.99, 'w');
yline(415.30, 'w');
yticklabels({'Dâ™ - 277.18','Eâ™ - 311.13', 'Gâ™ - 369.99', 'Aâ™ - 415.3'})
ylabel('Musical Note and Frequency (Hz)');  xlabel('Time (s)');
% We set the y-limits/range on our spectragram to be within the audible
% range of the fundamental frequencies of an acoustic guitar.
set(gca,'ylim',[100, 500],'Fontsize', 14);


%% Part #2 - Explore Pink Floyd - Comfortably Numb (Bass)
close all;
time_step = 0.1;  % Modify time step here.
tau_floyd = 0:time_step:L_floyd;
a_floyd_bass = 100;  % Modify width here.

ygt_spec_floyd_bass_gabor = zeros(length(tau_floyd), n_floyd);
ygt_spec_floyd_bass = zeros(length(tau_floyd), n_floyd);
% Save the bass notes for Part #3.
floyd_bass_notes = zeros(1, length(tau_floyd));
% Frequencies determined by looking at spectragram.
fund_freq_bass = [82.407, 92.499, 97.999, 110.0, 123.47];
for j = 1:length(tau_floyd)
    % Using a Gaussian Window Filter.
```

```
    g_bass = exp(-a_floyd_bass*(t_floyd-tau_floyd(j)).^2);
    yg_bass = g_bass.*y_floyd;
    ygt_bass = fft(yg_bass);
    ygt_spec_floyd_bass_gabor(j, :) = abs(fftshift(ygt_bass));

    % Find maximum frequency for each time slice - this is the note played.
    [max_val, i] = max(abs(ygt_bass));
    % Make sure the note is within first harmonic to a certain degree of
    % error.
    err = 3;
    music_note = get_fundamental_freq(k_floyd(i), fund_freq_bass, err);
    if (music_note == 0)
        % Not perfect - but if get_fundamental_freq doesn't return we set
        % the music note for this time slice of this bass to be the
        % previous note.
        music_note = floyd_bass_notes(j-1);
    end
    % We know it is this note for the first few based on the spectragram.
    if (j == 1 || j == 2)
        % Hacky.
        music_note = fund_freq_bass(end);
    end
    floyd_bass_notes(j) = music_note;

    % Use a gaussian filter to filter around the fundamental frequency of
    % the bass note we determined above.
    gaussian_tau = 0.1;
    gaussian_filter = exp(-gaussian_tau*(k_floyd-music_note).^2);
    ygt_bass_filtered = gaussian_filter.*ygt_bass;
    ygt_spec_floyd_bass(j, :) = abs(fftshift(ygt_bass_filtered));
end

% Plot spectrogram (Gabor transform with Gaussian window).
figure();
pcolor(tau_floyd, ks_floyd, ygt_spec_floyd_bass_gabor.');
shading interp;
title({['Pink Floyd - Comfortably Numb' ]
    ['Spectragram (Bass) - Gabor Transform with Gaussian Window']
    ['a = ' num2str(a_floyd_bass) ' and \Delta\tau = ' num2str(time_step)]
    });
colormap(hot);
ylabel('Frequency (Hz)');  xlabel('Time (s)');
set(gca,'ylim',[0, 1400],'Fontsize', 14);

% Plot the Spectrogram of the filtered bass line.
figure();
pcolor(tau_floyd, ks_floyd, ygt_spec_floyd_bass.');
shading interp;
title({['Pink Floyd - Comfortably Numb' ]
```

```
    ['Spectragram (Bass) - Clean']
    });
colormap(hot);
ylabel('Musical Note and Frequency (Hz)');  xlabel('Time (s)');
yticks([82.407, 92.499, 97.999, 110.0, 123.47]);
yticklabels({'E - 82.407', 'Gâ™ - 92.499', 'G - 97.999', 'A - 110.0', 'B -
123.47'});
yline(82.407, 'w'); yline(92.499, 'w'); yline(97.999, 'w');
yline(110.0, 'w'); yline(123.47, 'w');
% We set the y-limits/range on our spectragram to be within the audible
% range of the fundamental frequencies of a bass.
set(gca,'ylim', [60, 160],'Fontsize', 14);


%% Part #3 - Explore Pink Floyd - Comfortably Numb (Guitar)
a_floyd_guitar = 1000;  % Modify width here, we use the same time_step in
                        % Part #2.
ygt_spec_floyd_guitar_gabor = zeros(length(tau_floyd), n_floyd);
ygt_spec_floyd_guitar = zeros(length(tau_floyd), n_floyd);
floyd_guitar_notes = zeros(1, length(tau_floyd));  % Save the guitar notes.

% First we want to remove the bass line from the song along with its
% overtones from the data.
freq_dat = fftshift(fft(y_floyd));  % Transform to freq space.
err = 3;  % Specified error to remove frequency.
for j = 1:length(floyd_bass_notes)
    freq = floyd_bass_notes(j);  % Bass note to remove.
    if (freq == 0)
        continue;
    end
    % First harmonic.
    freq_dat(abs(freq_dat) <= (freq - err) & abs(freq_dat) >= ...
      (freq + err)) = 0;
    % Second harmonic.
    freq_dat(abs(freq_dat) <= (2*freq - err) & abs(freq_dat) >= ...
      (2*freq + err)) = 0;
    % Third harmonic.
    freq_dat(abs(freq_dat) <= (3*freq - err) & abs(freq_dat) >= ...
      (3*freq + err)) = 0;
    % Fourth harmonic.
    freq_dat(abs(freq_dat) <= (4*freq - err) & abs(freq_dat) >= ...
      (4*freq + err)) = 0;
    % Fifth harmonic.
    freq_dat(abs(freq_dat) <= (5*freq - err) & abs(freq_dat) >= ...
      (5*freq + err)) = 0;
    % Sixth harmonic.
    freq_dat(abs(freq_dat) <= (6*freq - err) & abs(freq_dat) >= ...
      (6*freq + err)) = 0;
end
```

```
% Transform back to time space to do Gabor.
filtered_floyd_dat = ifft(ifftshift(freq_dat));

% Do Gabor + Gaussian filter on cleaned data.
for j = 1:length(tau_floyd)
    g_guitar = exp(-a_floyd_guitar*(t_floyd-tau_floyd(j)).^2);
    yg_guitar = g_guitar.*filtered_floyd_dat;
    ygt_guitar = fft(yg_guitar);
    ygt_spec_floyd_guitar_gabor(j, :) = abs(fftshift(ygt_guitar));

    % Find maximum frequency for each time slice - this is the note played.
    [~, i] = max(abs(ygt_guitar));
    floyd_guitar_notes(j) = abs(k_floyd(i));

    % Use a gaussian filter to filter around the maximum frequency.
    gaussian_tau = 0.1;
    gaussian_filter = exp(-gaussian_tau*(k_floyd-floyd_guitar_notes(j)).^2);
    ygt_guitar_filtered = gaussian_filter.*ygt_guitar;
    ygt_spec_floyd_guitar(j, :) = abs(fftshift(ygt_guitar_filtered));
end

% Plot spectrogram (Gabor transform with Gaussian window).
figure();
pcolor(tau_floyd, ks_floyd, ygt_spec_floyd_bass_gabor.');
shading interp;
title({['Pink Floyd - Comfortably Number' ]
    ['Spectragram (Guitar) - Gabor Transform with Gaussian Window']
    ['a = ' num2str(a_floyd_guitar) ' and \Delta\tau = ' num2str(time_step)]
    });
colormap(hot);
ylabel('Frequency (Hz)');  xlabel('Time (s)');
set(gca,'ylim',[300, 1000],'Fontsize', 14);

% Plot the Spectrogram of filtered guitar.
figure();
pcolor(tau_floyd, ks_floyd, ygt_spec_floyd_guitar.');
shading interp;
title({['Pink Floyd - Comfortably Number' ]
    ['Spectragram (Guitar) - Clean']
    });
colormap(hot);
ylabel('Musical Note and Frequency (Hz)');  xlabel('Time (s)');
yticks([329.63, 369.99, 493.88, 587.33]);
yticklabels({'E - 329.63', 'Gâ™ - 369.99', 'B - 493.88', 'D - 587.33'});
yline(329.63, 'w'); yline(369.99, 'w'); yline(587.33, 'w'); yline(493.88, 'w');
set(gca,'ylim',[300, 700],'Fontsize', 14);


%% Bonus: Play Pink Floyd Sample Without Bass Notes
```

```
p8_floyd_guitar = audioplayer(filtered_floyd_dat, Fs_floyd);
playblocking(p8_floyd_guitar);
```

get_fundamental_freq.m

```
% Helper function for getting the fundamental frequency of note. This
% function is used to help filter out the overtones in our music data in
% the frequency domain.
% Takes in a note, array of fundamental frequencies in the first harmonic
% to compare note too, and a specified error to compare with.
function music_note = get_fundamental_freq(note, fund_freqs, err)
   music_note = 0;
   for i = 1:length(fund_freqs)
       fund_freq_lower = fund_freqs(i) - err;
       fund_freq_higher = fund_freqs(i) + err;
       % First harmonics
       if ((fund_freq_lower <= note) && (note <= fund_freq_higher))
           music_note = fund_freqs(i);

       % Second Harmonic
       elseif ((fund_freq_lower <= note/2) && (note/2 <= fund_freq_higher))
           music_note = fund_freqs(i);

       % Third Harmonic
       elseif ((fund_freq_lower <= note/3) && (note/3 <= fund_freq_higher))
           music_note = fund_freqs(i);

       % Fourth Harmonic
       elseif ((fund_freq_lower <= note/4) && (note/4 <= fund_freq_higher))
           music_note = fund_freqs(i);

       % Fifth Harmonic
       elseif ((fund_freq_lower <= note/5) && (note/5 <= fund_freq_higher))
           music_note = fund_freqs(i);

       % Sixth Harmonic
       elseif ((fund_freq_lower <= note/6) && (note/6 <= fund_freq_higher))
           music_note = fund_freqs(i);

       end
   end
end
```