AMATH 482- Computational Methods in Data Analysis: Filtering Frequencies from Music

Gargi Kher

February 7th, 2020

Abstract: Mathematical filters are used to analyze signals from sound or images. These functions make a big difference when it comes to removing background noise from such data. There are many different types of filter functions that target frequencies in different ways. Depending on the application, one filter might be more beneficial than the other. The Gabor transform is one type of filter that can be used to analyze sound. I will compare the Gabor filter to two other types of filter functions and describe how I used it to obtain the notes played from two short recordings of a song.

Sec. I. Introduction and Overview

In the first part of this project, I analyzed a short recording of Handel's *Messiah*, which was already incorporated into MATLAB. Applying the Gabor filter to the signal produced by the song, I plotted a spectrogram in order to determine the conditions that produced the best time and frequency resolution. In addition to the Gabor filter, I implemented two other wavelets to compare the resolutions they produced. These were the Mexican Hat Wavelet and the Mother Wavelet. The second part of the project required using the Gabor filter to denoise a recording of *Mary had a Little Lamb,* played by a piano and a recorder. The goal was to determine the notes being played by determining the central frequencies at each time.

For each of the parts, code had been given by the instructor to help us get started.

Sec. II. Theoretical Background

The Fourier Transform provides a convenient way to denoise data, as it can be applied in many ways to help clean up a signal. The Gabor filter (Eqn. 1) utilizes the Fast Fourier Transform (Eqn. 2) and allows us to gather as much information about a given piece of data as we can. Doing a Time Series analysis of a signal only allows us to get data in the time domain, with no frequency resolution. Conversely, the Fourier Analysis gives us most of the frequency data but no time data. The Gabor transform can be thought of as the middle ground, giving us some data in both of frequency and time domains.

$F(\tau, w) = \int_{-\infty}^{\infty} f(t)g(t-\tau)e^{-iwt}\,dt$ (1) The Gabor Transform

$F(k) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} e^{-ikx} f(x)\,dx$ (2) The Fast Fourier Transform

The Gabor filter, not surprisingly, is also known as the short-time Fourier Transform. Just looking at the equations shows similarities between the two. The Gabor filter is represented by a time (t) but unlike the FFT, is also represented by a frequency (w). Both functions represent the starting noisy function as a bunch of sines and cosines (given by the terms e^{-iwt} and e^{-ikt}, both complex values that can be written in terms of sines and cosines). The Gabor transform is just multiplying the FFT by some function $g(t-\tau)$ that fixes the filter around τ, better thought of as

a certain time/width. Thus, just like a simple Gaussian filter, changing the value of τ allows us to move the filter over the entire signal, and because the Gabor filter gives us information in both frequency and time domains, we can plot the information in a spectrogram showing frequency versus time.

When implementing the Gabor filter into MATLAB, we end up discretizing the function in the time and frequency domains. Thus, it is written as such:

$f(t) = e^{-a(t-\tau)^2}$ (3) MATLAB implementation of Gabor Transform

Where a determines the width of the filter- the smaller the a, the larger the filter. τ represents the distance from the middle of one filter to the next. The smaller the τ, the more overlap between filters there is. Because we are moving this filter over our entire data, we do want some overlap. It is important in practice to choose an a and τ such that the filter exclude frequencies of certain lengths as it moves through time.

Another strategy to filter data across a certain time interval is to use functions called wavelets. Wavelets also give time and frequency information like the Gabor transform, but they start by extracting the low frequencies (poor time resolution) and then break those up by reducing the filter window to extract higher frequencies (better time resolution). While there are many other wavelets and filters that can be implemented to filter data, I used just two in the first part: The Mexican Hat Wavelet and a Mother Wavelet. The Mexican Hat Wavelet (Eqn. 4) is expressed as a function of t, which is the length of time over which our song is recorded.

$f(t) = (1 - t^2)e^{\frac{-t^2}{2}}$ (4) Mexican Hat Wavelet

$\varphi_{a,b}(t) = \frac{1}{\sqrt{a}}\varphi\left(\frac{t-b}{a}\right)$ (5) Mother Wavelet

The Mexican Hat Wavelet is very similar to the Gabor filter, it just has a different shape. Unlike the Gabor, is wavelet doesn't depend on a, it just depends on t. The Mother Wavelet function (Eqn. 5), or the equation from which many other wavelets are derived, is more generic. This function represents any type of translation or rotation, and φ can represent any wavelet. The variables a and b are constants, where changing a can stretch the function horizontally and vertically, and b represents the interval of time the recording is playing on.

Because we were obtaining frequency and time data for the same recording, many of the results had to be graphed in plots called spectrograms. Spectrograms plot time on the x-axis and frequency on the y-axis. The idea is we should be able to see the frequency associated with every time (which was the purpose of applying these filter functions to the data). Consistent with the Heisenberg Uncertainty Principle, however, the more time resolution you get for an object, the less frequency resolution you receive and vise-versa. Thus, spectrograms can't tell us everything about the time and frequency. Once implemented, the filter function must be modified so we can obtain the best possible resolution of a spectrogram.

Sec. III. Algorithm Implementation and Development

Part 1

I first loaded Handel's *Messiah* into MATLAB using the instructor-given code. In order to implement a Gabor function or any wavelet, I had to discretize the signal. I defined L as the length of the song (9 seconds). This value was used to help construct the time domain, discretized into n points. Normally, for a FFT, the value of n would have to be a power of two. This time, because we were given the length of the signal in the starter code, I defined n to be this length. When defining my vector of frequencies, I had to account for the fact that n was odd. Thus, instead of the first half of k going from $0: n/2 - 1$, it had to go from $0: n/2$. I then constructed a shifted version of the frequency vector for ease of plotting in future steps (Appendix B: Lines 1-18).

I implemented the Gabor filter, Mexican Hat Wavelet, and Mother Wavelet in different sections, but most of my code was the same. Because I was testing for different values of a for Gabor and Mother Wavelet, I had these in a nested for loop. The outside iterated over the values of a that were in the vector I constructed. The inside iterated over the length of τ , or the overlap of my filter function. I ended up choosing the values for a and τ I thought were best after running the code a few times. Then, multiplied the signal by this filter function, and stored the absolute value of the FFT of the result in a vector. The idea was that I was storing the time and frequency data corresponding to each value in τ, which came together to comprise the entire length of the recording. I graphed this information in a spectrogram, which gave me the resolution between time and frequency (Appendix B: Lines 19-38, 39-55, 56-75).

Part 2

We were given starter code for both the piano and recorder data. After loading the starter code, I discretized the interval just like in Part 1, using n to represent the length of the signal. This time, because n was even, I didn't have to change how I constructed k (Appendix B: Lines 77-98 and 127-146). I started by defining τ ($tslide$) and a for my Gabor filter. These were determined by trial and error, but the general idea was to make sure that a was big (small filter) and τ was small (more overlap). To ensure the filter would move over the entire time interval, I constructed my filter function inside of a for loop that iterated over τ (Appendix B: Lines 99-104, and 146-151). Once the Gabor filter was applied, it was necessary to find the central frequencies.

The central frequency had to be found every time the filter was applied, as it corresponded to the note that was played at that time. I found the index at which the absolute value of the filtered FFT function ($St1$ and $St2$) was maximum and plugged that index into k to find the actual central frequency value at that time interval. All of the central frequencies were stored in a vector ($vector\_piano$ and $vector\_rec$), which I used to determine the note being played either on the piano or recorder using the chart we were given. When constructing the spectrogram, I made sure to store the Gabor filtered function in a vector (St_spec1 and St_spec2) Because this matrix would have been too big to plot (around 60000 or 70000 elements, depending on the instrument), and we only care about the central frequencies, I reduced this matrix to correspond to the indexes of ks that represent the central frequencies. I

divided each frequency by 2π to convert to Hertz. I was then able to plot the spectrograms for each of the recordings (Appendix B: Lines 104-124 and 151-171).

Sec. IV. Computational Results

Part 1

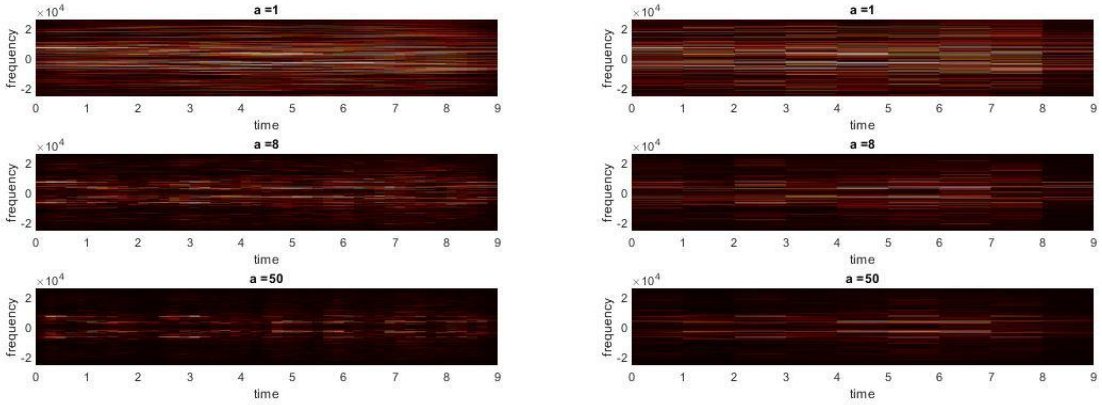

Figure 1: Gabor Filter with different a-values for τ = 3 (left) $\tau = 0.1$ (right).

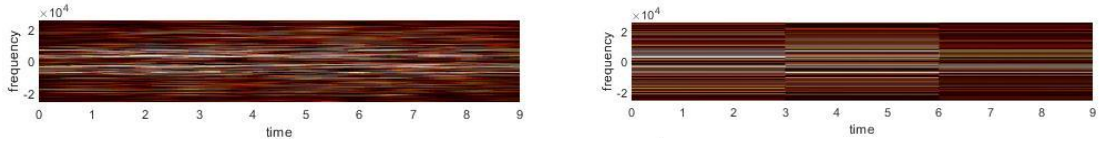

Figure 2: Mexican Hat Wavelet with $\tau = 0.1$ (left) and $\tau = 3$ (right).

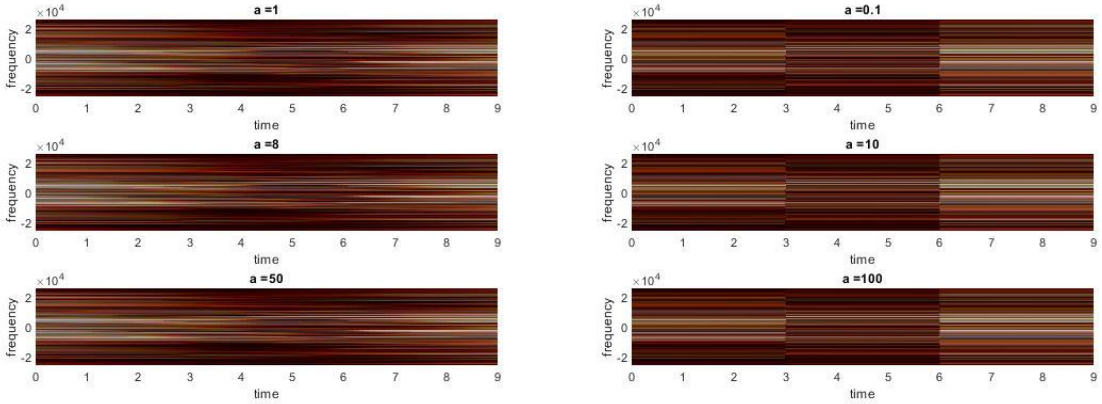

Figure 3: Mother Wavelet with different a-values for τ = 0.1 (left) and τ = 3 (right).

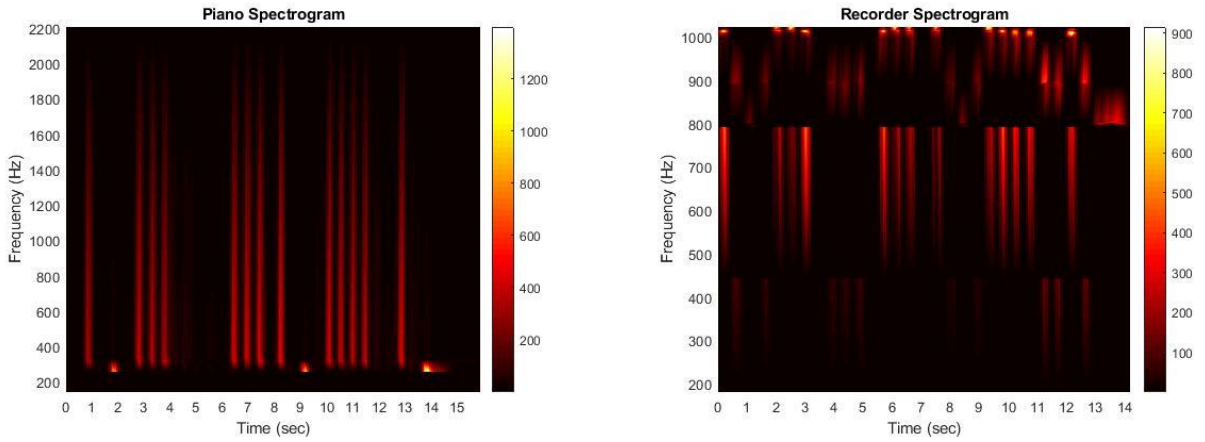

Figure 4: Unfiltered frequencies for the piano and recorder recordings (after Gabor transform).

Sec. V. Summary and Conclusions

Part 1

As mentioned earlier, there was a lot of trial and error for this part. For the Gabor Filter (Figure 1, Sec. IV.), I tried multiple a values until I found the one that gave the best resolution of time and frequency, which I determined to be $a = 8$. Figure 1 shows the difference between the windows for a larger τ and smaller τ. Setting τ = 0.1 allowed more overlap of the filters and created more defined windows with better frequency and time resolution.

Because the Mexican Hat Wavelet (Figure 2, Sec. IV.) only depended on time, I only had to manipulate τ. I found that as τ increased, the time resolution became better. Even though the frequency resolution remained about the same, I started to see wider windows as I increased τ. This is likely due to increasing the width of time we are looking over. Compared to a τ of 0.1, this filter displays better time resolution for a τ of 3.

Lastly, for the Mother Wavelet (Figure 3, Sec. IV.), I noticed that changing a didn't affect it as much. Just like the Mexican Hat Wavelet, its time resolution improved as τ increased. Compared to a τ of 0.1, the time domain is better defined with a τ of 3. It is worth noting that I assumed a φ of 1 for this wavelet, so that my equation was just $\varphi_{a,b}(t) = \frac{1}{\sqrt{a}}\left(\frac{t-b}{a}\right)$. Mother Wavelets for different functions might produce different results than mine.

Part 2

Piano score: C/C/G/B/B/D/E/D/Middle C/C/E/D/E/D/Middle C/D/E/D/E/D/E/Middle C/E
Recorder score: C/A/G/A/C/G/A/C/A/C/A/G/A/C/A/C/A/G
I obtained the notes by looking at the vector of central frequencies I constructed and comparing those frequencies to the notes on the given chart. Based on the spectrograms (Figure 4, Sec. IV.), which were not filtered after applying the Gabor Transform, we see these central frequencies being played at the brightest points. The lines directly above our central frequencies are the overtones. They're the noise that occurs when the note of interest is played, but when depicted in a spectrogram we can easily see that the overtones are not the same frequency as our note of

interest. The spectrograms for both recordings seem to be spaced similarly, which makes sense because they both represent the same song. The lines that are closer together represent shorter notes. Lines that are spaced further apart are notes that have been held for a longer time.

Sec. VI. Sources/References

Kutz, Nathan J. (2013). *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data*

Appendix A. MATLAB functions used and brief implementation explanation

$pcolor$ – plots a function of two variables, contains features like colorbar that allow for the identification of certain values based on the intensity of the color presented.
$find$ – finds the index(es) at which a specified condition occurs for a given function.
$audioread$ – loads data from a .wav file, outputting the signal and sample rate.
$audioplayer$ – stores the signal and sample rate of an audio file.
$playblocking$- plays an audiofile.
$subplot(x, y, z)$- similar to the plot function, but splits multiple plots into one figure, where x is the number of rows, y is the number of columns, and z is the position of each plot.

Appendix B. MATLAB codes

```
%%Part 1
%% instructor given code
load handel
v = y';
plot((1:length(v))/Fs,v);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Signal of Interest, v(n)');
% plays the music
p8 = audioplayer(v,Fs);
playblocking(p8);
%% Discretizing
L = 9;
n = 73113; %chose my own n
t2 = linspace(0,L,n+1);
t = t2(1:n);
k = ((2*pi)/L)*[0:n/2 -n/2:-1]; %modified k
ks = fftshift(k); %shifting so we don't have to shift when plotting
%% Gabor filter
a = [1 8 50];
tau = 0:0.1:L;
Sgt_spec = zeros(length(tau),n);
for j = 1:length(a)
for i = 1:length(tau)
    g = exp(-a(j)*(t-tau(i)).^2); %Gabor filter
    Sg = g.*v; %applying filter to signal
    Sgt = fft(Sg); %taking FFT of filtered signal frequency
    Sgt_spec(i,:) = fftshift(abs(Sgt));
end
subplot(length(a),1,j)
colorbar
```

```
pcolor(tau,ks,Sgt_spec.')
shading interp
colormap(hot)
title(['a =',num2str(a(j))])
xlabel('time')
ylabel('frequency')
end
%% Mexican Hat Wavelet
tau = 0:3:L;
Sgt_spec = zeros(length(tau),n);
for i = 1:length(tau)
    g = (1-(t-tau(i)).^2).*exp((-(t-tau(i)).^2)/2); %Mexican Hat wavelet
    Sg = g.*v; %applying filter to signal
    Sgt = fft(Sg); %taking FFT of filtered signal frequency
    Sgt_spec(i,:) = fftshift(abs(Sgt));
end
subplot(length(a),1,j)
colorbar
pcolor(tau,ks,Sgt_spec.')
shading interp
colormap(hot)
title(['a =',num2str(a(j))])
xlabel('time')
ylabel('frequency')
%% Mother Wavelet
a = [0.1 10 100];
tau = 0:3:L;
Sgt_spec = zeros(length(tau),n);
for j = 1:length(a)
for i = 1:length(tau)
    g = 1/sqrt(a(j)).*((t-tau(i))./a(j)); %Mother wavelet function
    Sg = g.*v; %applying filter to signal
    Sgt = fft(Sg); %taking FFT of filtered signal frequency
    Sgt_spec(i,:) = fftshift(abs(Sgt));
end
subplot(length(a),1,j)
colorbar
pcolor(tau,ks,Sgt_spec.')
shading interp
colormap(hot)
title(['a =',num2str(a(j))])
xlabel('time')
ylabel('frequency')
end
%% Part 2
%% Music 1- Piano
clc;clear all;
% Instructor given code
[y,Fs] = audioread('music1.wav'); %y= sampled data (m=# of audio samples
read, n=#of audio channels in file), Fs= sample rate(Hz)
tr_piano=length(y)/Fs; % record time in seconds
figure(1)
subplot(2,1,1)
plot((1:length(y))/Fs,y);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Mary had a little lamb (piano)');
```

```
p8 = audioplayer(y,Fs);
playblocking(p8);
%% Discretizing
sig = y.';
L = tr_piano;
n = length(y); %chose my own n
t2 = linspace(0,L,n+1);
t = t2(1:n);
k = ((2*pi)/L)*[0:n/2-1 -n/2:-1];
ks = fftshift(k); %shifting so we don't have to shift when plotting
%% Creating Spectrogram
tslide = 0:0.1:L;
a1 = 100;
vector_piano = zeros();
ind_piano = zeros();
St_spec1 = zeros(length(tslide),n);
for b = 1:length(tslide)
g1 = exp(-a1*(t-tslide(b)).^2);
St1 = fft(g1.*sig);%St1 is in the frequency domain
St_spec1(b, :) = fftshift(abs(St1));
    %find max at each point
    k0 = abs(max(St1));
    indice = find(abs(St1)==k0);
    ksindexes = find(abs(ks)==k(indice(1)));
    vector_piano(:,b) = abs(ks(ksindexes(1))./(2*pi)); %convert to Hz
    ind_piano(:,b) = ksindexes(2);
end
St_spec1 = St_spec1(:,[ind_piano]);
%% Plotting
pcolor(tslide,vector_piano,St_spec1.') %unfiltered function
shading interp
colormap(hot)
colorbar
title('Piano Spectrogram')
xlabel('Time (sec)')
ylabel('Frequency (Hz)')
set(gca,'XTick',[0:1:L])
%% Music 2- Recorder
clear;clc;
[y,Fs] = audioread('music2.wav');
tr_rec=length(y)/Fs; % record time in seconds
figure(2)
subplot(2,1,1)
plot((1:length(y))/Fs,y);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Mary had a little lamb (recorder)');
p8 = audioplayer(y,Fs);
playblocking(p8);
%% Discretizing
sig = y.';
L = tr_rec;
n = length(y); %chose my own n
t2 = linspace(0,L,n+1);
t = t2(1:n);
k = ((2*pi)/L)*[0:n/2-1 -n/2:-1];
ks = fftshift(k); %shifting so we don't have to shift when plotting
```

```
%% Creating spectrogram
tslide = 0:0.1:L;
a2 = 100;
vector_rec = zeros();
ind_rec = zeros();
St_spec2 = zeros(length(tslide),n);
for b = 1:length(tslide)
g2 = exp(-a2*(t-tslide(b)).^2);
St2 = fft(g2.*sig); %St1 is in the frequency domain
St_spec2(b, :) = fftshift(abs(St2));
    %find max at each point
    k0 = abs(max(St2));
    indice = find(abs(St2)==k0);
    ksindexes = find(abs(ks)==k(indice(1)));
    vector_rec(:,b) = abs(ks(ksindexes(1))./(2*pi)); %convert to Hz
    ind_rec(:,b) = ksindexes(2);
end
St_spec2 = St_spec2(:,[ind_rec]);
%% Plotting
pcolor(tslide,vector_rec,St_spec2.') %unfiltered function
shading interp
colormap(hot)
colorbar
title('Recorder Spectrogram')
xlabel('Time (sec)')
ylabel('Frequency (Hz)')
set(gca,'XTick',[0:1:L])
```