AMATH 582 Homework 2

Shabab Ahmed
Github link: https://github.com/sahmed95/AMATH582

February 7, 2020

Abstract

This project deals with time-frequency analysis of music signals. The first part of the project involves analyzing a portion of Handel's Messiah. We produce spectograms of the piece of music using Gábor transforms with the standard Gaussian window. We explore the effect of window width of the Gábor transform on the time-frequency analysis and also consider spectograms produced by using different Gábor windows, namely, the Shannon window and the Mexican hat wavelet. We also look at the tradeoff between temporal and spectral resolution and its relation to over and undersampling. For the second part, we investigate the spectral differences of a piano versus a recorder using audio files of 'Mary had a little lamb' on the two instruments. We recreate musical scores for each rendition of 'Mary had a little lamb' by producing spectograms using Gábor transform with the Gaussian window.

1 Introduction and Overview

Fourier transform is a fundamental method of analyzing signals. However, Fourier transform can only capture the frequency content of the signal and fails to consider the moment in time when the different frequencies actually occurred. Music signals will usually comprise of different frequency components that occur at different times. The average Fourier components of a song changes in time and it will have high and low notes. The high notes will consist of high frequency components and the low notes will consist of low frequency notes. Fourier transform will give us all the frequency components but fail to tell us when they occurred. It will be difficult to recreate the music signal if we do not have information on when these high/low pitches (high/low frequencies) occur at what time. Therefore, Fourier analyis will not be sufficient to analyze a piece of music and therefore we look for modifications so that we can achieve both time and frequency information.

We use a modification of the Fourier transform (Gábor transform) in this project. Essentially, we create a time window and we take the Fourier transform of the signal in that time window and slide that time window across the signal. This should help us gather frequency information at different instances in time. This modification of the Fourier transform is described in more detail in the Theoretical background section.

This project consists of looking at different audio signals in two parts. For the first part of the project, we look at a 9 second piece of Handel's Messiah. We use time-frequency analysis to reproduce the musical signal. We want to extract both temporal and spectral information so that we know what frequencies were exhibited in the signal along with the time in which they were exhibited using the modified Fourier transform. We also play around with different parameters, width of the time window and sampling rate, to see the effect of these parameters on the spectogram. This relates to the trade off we mentioned in the abstract. Gábor transform will take away accuracy in both time and frequency domain to give information about both of them simultaneously. Using a small window will mean that long frequencies will not appear in the window and so will not be present in the spectogram. Alternatively, using a large window would mean that we lose resolution in the time domain in exchange of getting better resolution in the frequency domain.

On the second part of the project, we look at audio signals produced by a recorder and a piano of the song 'Mary had a little lamb'. Using Gábor transform, we attempt to reproduce the musical score of the song and analyze the differences in the signal produced by studying the spectogram. In what follows, we give a brief overview of the process of reproducing the musical scores.

The technique for reproducing the music score will be the same for both the parts so we give a single overview for the two parts. We start off by loading the Handel piece and by loading the .wav files containing the audio signal of 'Mary had a little lamb'. Then, we divide the signal by 2 and calculate the length of the signal (audio vector) and the sampling rate. We have to ensure that the the audio vector is of even length because the Fourier transform uses 2^n points. The Handel piece has an audio vector of even length and so we leave out the endpoint. This is because

the first and last values of the audio vector is the same (= 0) and the Fourier transform assumes periodic signals so we do not need the information from the last point. Also, the audio vector created by MATLAB is a column vector and we need to transpose it to a row vector because it will be necessary for applying the filters. This determines the number of Fourier modes. After this step, we identify the last moment in time and then discretize the time domain into a vector with equally spaced points. We scale the wavenumbers by $\frac{2\pi}{\mathtt{L}}$ where `L` is the length of the time interval since Fourier transform assumes a 2π periodic signal. We also shift the wavenumbers.

We move onto taking the Gábor transform of the signal. We start off by defining a time slide vector which consists of equally spaced points in time. The spacing of the time slide vector determines whether the signal will be over or undersampled. We also create a parameter that will define the width of the Gábor window. Finally, we intialize the matrix that will produce the spectogram. For each point in the time slide vector, we create a filter centered at that point with width as mentioned above. There are different kinds of Gábor windows available and we test the Gaussian window, Mexican hat wavelet and the Shannon window in the first part of the project. For the second part, we use the Gaussian window only. After creating the filter, we multiply the audio signal with the filter and then take the Fourier transform of the product. We take the absolute value and then shift the result of the previous process before appending it as a row in the matrix that will produce the spectogram. After this process, we produce the spectogram of the audio signal by using the `pcolor` function in MATLAB.

Spectograms are going to be an important part of visualizing our results. Hence, we provide a brief explanation of the spectogram in the subsection below. The rest of the paper is structured as follows: Section 2 contains the Theoretical Background, Section 3 provides an outline of the algorithm used and Section 4 describes the results. Finally, Section 5 concludes.

1.1 Spectogram

A spectogram is a visual way of representing the signal strength of a signal over time by producing a graph of the spectrum of frequencies present in the signal as it varies with time. In this project, a spectogram will be a two-dimensional graph with time being represented on the x-axis and the frequency being represented on the y-axis. The graph is a color map and the amplitude of a particular frequency at a particular time is represented by the intensity of color at that point. Therefore, spectograms can be thought as the fingerprint of sound and is often used in music detection software.

2 Theoretical Background

2.1 Gábor Transform

As we learned from our textbook [1], Fourier introduced the concept of representing a given function $f(x)$ by a trigonometric series of sines and cosines:

$$f(x) = \frac{a_0}{2} + \sum_{i=1}^{\infty} (a_n \cos nx + b_n \sin nx) \quad x \in (-\pi, \pi]. \tag{1}$$

This led to the idea of Fourier transform we have used previously to obtain spectral information about a given signal. However, according to our discussion above we cannot directly use the Fourier transform to recreate musical scores because we would need frequency information in time and the Fourier transform fails to localize a signal both in time and frequency domains [1].

Therefore, we need to consider modifications of the Fourier transform. Gábor Dénes provided a formal method for localizing both time and frequency. He modified the Fourier transform kernel to introduce the kernel:

$$g_{t,\omega}(\tau) = e^{i\omega\tau} g(\tau - t) \tag{2}$$

where the new term to Fourier kernel $g(\tau - t)$ aims to localize both time and frequency [1]. The *Gábor transform* is then defined as the following:

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

The function $g(\tau - t)$ acts as a time filter for localizing the signal over a specific window of time. The integration over the τ slides the time filtering window down the entire signal so that we achieve spectral information at each instant of time. A pictoral representation of the Gábor transform extracting time-frequency content of a signal along with some important mathematical properties of the Gábor transform is provided in [1] and is not discussed here.

2.2 Discrete Gábor Transform

In practice, Gábor transform is computed by discretizing the time and frequency domain. By discretizing, the transform is done on a lattice of time and frequency. Let us consider the lattice,

$$v = m\omega_0 \tag{4}$$

$$\tau = nt_0 \tag{5}$$

where m and n are integers and $\omega_0, t_0 > 0$ are constants. The discrete version of $g_{t,\omega}$ becomes

$$g_{m,n}(t) = e^{i2\pi m\omega_0 t} g(t - nt_0) \tag{6}$$

and the Gábor transform becomes

$$\bar{f}(m,n) = \int_{-\infty}^{\infty} f(t)\bar{g}_{m,n}(t)dt = (f, g_{m,n}) \tag{7}$$

We can notice that if $0 < t_0, \omega_0 < 1$ then the signal is oversampled and time frequency yields good localization of the signal in both time and frequency. If $w_0, t_0 > 1$, the signal is undersampled and the Gábor lattice is incapable of reproducing the signal. This is the tradeoff we discussed above and will be explored further in the results. Given that the Heisenberg relationship must hold, the shorter the time filtering window the less information about frequency and more information about time is obtained and vice versa. This is also demonstrated via a graphical description in Figure 118 of [1].

2.3 Gábor Window

Several windows can be used as filters for the Gábor window. We consider the following windows in this project:

2.3.1 Gaussian window

The simplest Gábor window to implement is a Gaussian time-filter centered at some time τ with width a. As mentioned above, a is going to effect the resolution of the time and frequency domains. The Gábor window with respect to the Gaussian filtering is given by:

$$g_{a,b}(t) = e^{-a(t-b)^2} \tag{8}$$

where a is the width parameter and b is the translation parameter which we have been referring to as τ thus far.

2.3.2 Mexican Hat Wavelet

This is one of the more common wavelets. It is essentially a second moment of Gaussian in the frequency domain. The definition of this wavelet and its transform are as follows:

$$\psi(t) = (1 - t^2)e^{-\frac{t^2}{2}} = -\frac{d^2}{dt^2}e^{-\frac{t^2}{2}} = \psi_{1,0} \tag{9}$$

$$\hat{\psi}(\omega) = \hat{\psi}_{1,0}(\omega) = \sqrt{2\pi}\omega^2 e^{-\frac{\omega^2}{2}} \tag{10}$$

Eq (9) gives the wavelet for $a = 1$ and $b = 0$.The Mexican hat wavelet can be dilated and translated easily. We can define the general wavelet for dilation a and translation b as the following:

$$\psi(t) = \frac{1}{\sqrt{a}}(1 - (\frac{t-b}{a})^2)e^{-(\frac{t-b}{\sqrt{2}a})^2} \tag{11}$$

This wavelet has excellent localization properties in both and time frequency due to the minimal time-bandwidth product of the Gaussian function [1]. Some properties of the wavelets are discussed in [1].

2.3.3 Shannon Window

Shannon window is simply a step filter in the part of the domain we are interested in. The filter at the parts of the domain we are interested in is equal to 1 and the rest of the filter is 0. It is also referred to as a square wave and is given by the following:

$$g_{a,b}(t) = \begin{cases} 0 & \text{if } |\frac{t-b}{a}| > \frac{1}{2} \\ 1 & \text{if } |\frac{t-b}{a}| \leq \frac{1}{2} \end{cases} \tag{12}$$

where a is the width parameter and b is the translation parameter.

3 Algorithm Implementation and Development

In this section we lay out our steps to producing spectograms of the audio signals in an orderly fashion.

1. Use `load`/`audioread` command to either load the audio signal or read the `.wav` files containing the audio signals
2. Divide the signal (`y` by 2), take its transpose and calculate the length of the signal (`n`)
   (a) If the signal is of odd length, we take out the last point (in our case it is equal to the first point)
3. Identify the final time `L` using $\frac{\texttt{n}}{\texttt{Fs}}$, where `Fs` is sampling rate
4. Discretize the time domain into a vector (`t`) of evenly spaced (`n`) points
5. Compute wavenumbers (`k`), rescale them by $\frac{2\pi}{L}$ and shift them to get shifted wavenumbers (`ks`)
6. Define window width parameter (`a`)
7. Define translation parameter (`dt`) and create a vector (`tslide`) of time with each point being `dt` apart. This will slide the window across the signal
8. Initialize a matrix (`vgt_spec`) that will be fed into `pcolor` to produce the spectogram.
9. For each point in vector `tslide`, create the filter `g` centered around that point and of width `a`
   (a) The filters will be different for different windows. We use Gaussian, Mexican Hat wavelet and Shannon Window. Equations (8), (11) and (12) gives us the filter related to the windows. For Mexican Hat Wavelet and Shannon Window, we defined a function for the filter first and then feed into our loop.
10. Multiply the signal by the filter (`g`) and take the FFT of the product
11. Add the result of the FFT to the matrix (`vgt_spec`) as a row
12. Produce the spectogram using `tslide` as the x-co-ordinate, `ks` as the y-co-ordinate and `vgt_spec` specifies the colors
13. Repeat the process for different width parameters `a` and translation parameter `dt` as mentioned in the Computational Results section.

Part II:

1. Repeat parts 1-11 from the previous part.
2. This step is also the same but we use $\frac{ks}{2\pi}$ as our y-co-ordinate. This helps us use frequency in the time domain using the relationship

$$\omega = 2\pi f \tag{13}$$

where ω are the wavenumbers and f is frequency in Hertz.

4 Computational Results

Figure (1) contains the signal of interest, that is, the 9 second long audio signal of Handel's Messiah. The bottom panel of the figure contains the Fourier transform of the signal of interest. The top panel contains information only in time and the bottom panel contains information only in the frequency domain. Thus, we use Gabor transform to capture information in both the time and frequency domain.

We start off by trying the Gaussian window in the Gabor transform and producing the spectogram. Figure (2) contains information regarding this. The top panel contains the Gaussian filter in red, the middle panel contains the product of the signal and the bottom panel contains the FFT of the product with the results normalized. In the algorithm, we mentioned that we do this for for every point in our time slide vector which means that we are essentially sliding the red filter in the top panel across the entire signal and then repeating the process of taking a product and its FFT. Figure(2) is a visual representation of this for one specific point in the time vector.

We also tamper with the width of the filter and use the following values: $a = 500, a = 250, a = 50$ and $a = 2$. Figure(3) illustrates the spectogram produced using the Gaussian filter in increasing window width. We use a time step, $dt = 0.1$. In these figures, we have both negative and positive frequencies but we really care about the positive frequencies. The top left panel contains the spectogram produced by using width parameter $a = 500$ and $dt = 0.1$. We expect all the frequencies to be captured in this case because the signal is oversampled. The window is really narrow which gives a really good temporal resolution in exchange of poorer spectral resolution. We can see this in the sense that lower frequencies are hard to notice because their amplitudes are lower than the higher frequency components as the intensity of color indicates the amplitude. If we focus on the positive values (spectogram is symmetric) ,We can also distinguish two different bands of frequencies that are present and this corresponds to the men's and women's voices. We can see that for narrower window width, that is, greater a value , we get greater temporal resolution in exchange of frequency information. As we decrease the value of a, that is, increase the window width we get better frequency resolution as more frequencies show up in the spectogram. These are the low frequencies whose associated wavelengths can now fit into the window. We can see that in the case of $a = 2$, we have extremely poor temporal resolution because the window size is really wide. However, we do have lot more frequencies appearing in the spectogram. We know that if we keep decreasing a to the point that it the entire signal window size we will just have spectral information and that is what this points towards. The width paramater a equal to 250 seems to be a middle value and gives a good localization in both time and frequency.

Figure (4) contains the differences between oversampling and undersampling the signal. For the undersampling case, we take $dt = 0.5$ and $a = 250$. We can see that spectogram seems washed out and stretched in the x-direction. We do not have good time resolution because we cannot really tell when the frequencies occured. We are also unable to get a lot of frequencies in this case and only a handful of frequencies appear in the spectogram. On the other hand, we used $a = 250$ and $dt = 0.01$ for the oversampling case. The figure is qualitatively similar to the top right panel of Figure (3). This gives good localization of both time and frequency.

We move onto using the Mexican hat wavelet and the Shannon window to produce spectograms. Figure (5) contains the spectograms produced by these filters. We use $a = 0.05$ and $dt = 0.1$. These values of the parameter give good localization in both time and frequency. One interesting thing we noticed smaller values of a gave better results in the time domain. The spectogram for the Mexican hat wavelet for these parameters look similar to the oversampled spectrogam. Figure (6) contains the Shannon filter with the signal in the top panel. The bottom panel contains the spectogram produced using $a = 3$ and $dt = 0.1$. The spectogram seems to be really blurry and the temporal resolution is lost. This might be due to the sharp edges of the filter. It does not do a great job of reproducing the musical score.

For the second part, we look at the musical score of 'Mary had a little lamb'.Figure (7) contains the signals in the time and the frequency domain for the piano and the recorder. Figure(8) contains the spectogram produced from a piano using the Gaussian window and width parameter $a = 500$ and translation parameter $dt = 0.1$. We had to zoom in to see the spectogram more clearly so we only plot values in [0,950] Hz. We can see that the frequency components are about 260Hz, 290Hz and 320Hz. We are plotting frequency in Hertz and not the wavenumbers. This is easily done by using Equation (13).We can also notice overtones that occur at integral multiples of the fundamental frequencies, that is, the frequencies mentioned above. However, these are really faint implying that they have low amplitudes. This can be removed by filtering in the frequency domain but we do not implement this in this project because the musical score seems to be clean enough. 260 Hz refers to note middle C, 290 Hz refers to note D and 320 refers to note E. The musical score looks like: EDCDEEEDDDEEEEDCDEEEEDDEDC.

Figure(9) contains the signal from the same song using a recorder. The top panel contains the signal in the time domain and the bottom panel contains the signal in the frequency domain. Figure(10) contains the spectogram from the recorder. The frequency components have the same structure in the sense that they are distributed similarly to the piano and they occur at the same time as the piano. However, the frequency components are much higher. The

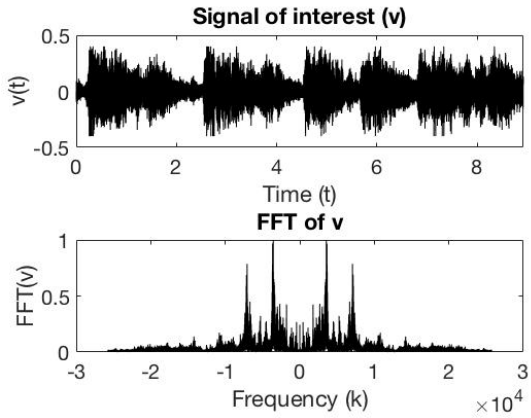

Figure 1: The top panel contains the signal of interest v and the bottom panel contains the Fourier transform of v

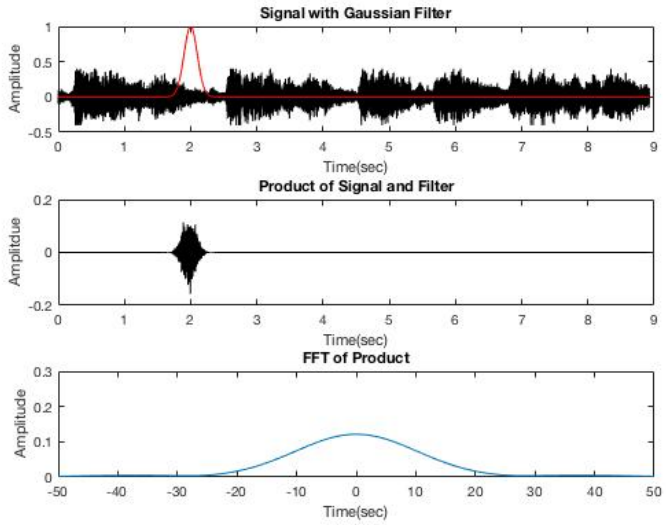

Figure 2: The top panel contains the signal of interest v in black and the Gaussian filter in red centered at some point and of width $a = 50$. The middle panel contains the product of the signal with the filter and the bottom panel contains the FFT of the product with the results normalized.

fundamental frequencies are around 810Hz, 920Hz and 1020 Hz. The overtones in this case are even fainter than the piano. They do happen in integral multiples of the fundamental frequencies mentioned. We can see them if we zoom in but otherwise it is hard to see. The recorder is much less noisy in that sense. 810 Hz corresponds to note G, 920 Hz corresponds to note A and 1020 Hz corresponds to B. These are next to each other on the piano just like E, D and C. The musical score follows the same pattern from above with B = E, A = D, and G = C.

5 Summary and Conclusions

Gabor transform gives a method whereby time and frequency can be simultaneously characterized. However, there are limitations to the method as has been illustrated by using different values for the width and translation parameters. Specifically, the method is limited by the time filtering itself. If we take a filter centered at τ and of width a, any portion of the signal with a wavelength longer than the window is lost. There is a trade off that occurs between time and frequency resolution as accuracy in one of these comes at the expense of accuracy in the other domain. Despite these limitations, this is a powerful method that lets us recreate musical scores and has applications in music detection software, for example. We also found the differences in the piano and the recorder. The piano seemed to be more noisy in terms of overtones. The frequency components of the recorder were higher than the frequency

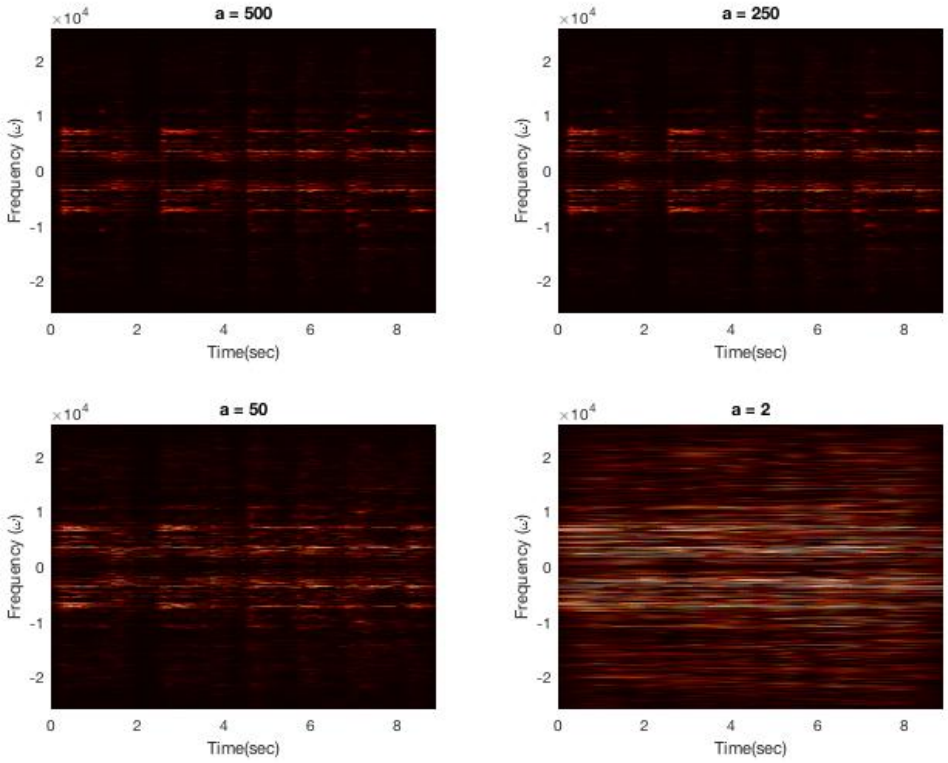

Figure 3: Spectograms of the signal using the Gaussian window with window width parameter 500, 250, 50 and 2

components of the piano.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data.* Oxford University Press, 2013.

Appendix A MATLAB Functions

This section contains some important MATLAB functions with a brief implementation explanation.

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `[y,Fs] = audioread(filename)` reads data from the file named `filename`, and returns sampled data, `y`, and a sample rate for that data, `Fs`. In our case, we actually calculated the sampling rate and so the function simply returned the sampled data.
- `load(filename)` loads data from `filename`.
- `Y = fft(X)` computes the discrete Fourier transform (DFT) of `X` using a fast Fourier transform (FFT) algorithm. In our case, the signal `X` is a vector and so `fft(X)` returns the Fourier transform of the vector.
- `Y = fftshift(X)` rearranges a Fourier transform X by shifting the zero-frequency component to the center of the array. `X` is a vector in our case and so `fftshift` swaps the left and right halves of `X`.
- `pcolor(X,Y, C)` creates a pseudocolor plot using the values in matrix `C`. A pseudocolor plot displays matrix data as an array of colored cells. `X` and `Y` specifies the x and y co-ordinates of the corners of the cells (vertices) and matrix `C` specifies the colors at the vertices.

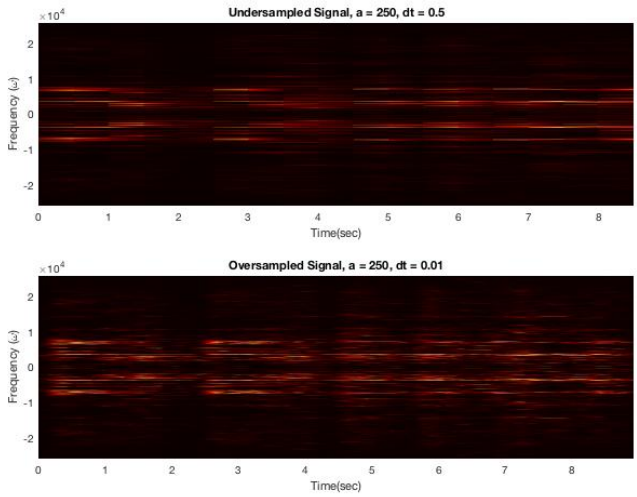

Figure 4: Undersampled vs Oversampled

Appendix B MATLAB Code

We provide the code used in the project.

```
% Shabab Ahmed
% HW 2 582
% Part 1

%-----------------------------------------------------------------------
clear all; close all; clc

% load signal

load handel

% take transpose of signal and divide by 2

v = y'/2;



% make signal even length
v= v(1:end-1);
n = length(v);
L = (n)/Fs; %final time
t = linspace(0,L,n); %time domain
k = (2*pi/L)*[0:n/2-1 -n/2:-1];  % make k 2pi periodic
ks = fftshift(k);                % shift

%-----------------------------------------------------------------------

% Plotting signal of interest

figure(1)
subplot(2,1,1) % Time domain
plot((1:length(v))/Fs,v, 'k')
xlim([0 t(end-1)])
```

```
set(gca, 'Fontsize', [20]),
xlabel('Time (t)'), ylabel('v(t)')
title('Signal of interest (v)')

% Fourier transform of signal
vt = fft(v);

subplot(2,1,2) %Fourier domain
plot(ks, abs(fftshift(vt))/max(abs(vt)),'k');
set(gca, 'Fontsize', [20])
xlabel('Frequency (k)');
ylabel('FFT(v)');
title('FFT of v')

%-----------------------------------------------------------------------

% Gabor filtering using Gaussian window

figure(2)


a = 500;
vgt_spec=[];
dt = 0.1;
tslide=0:dt:t(end-1);
for j=1:length(tslide)
    g=exp(-a*(t-tslide(j)).^2); % Gabor window
    vg=g.*v;                    % product of signal and filter
    vgt=fft(vg);
    vgt_spec=[vgt_spec; abs(fftshift(vgt))];
%     subplot(3,1,1), plot(t,v,'k',t,g,'r')
%     xlabel('Time(sec)'); ylabel('Amplitude')
%     title('Signal with Gaussian Filter')
%
%     subplot(3,1,2), plot(t,vg,'k')
%     xlabel('Time(sec)'); ylabel('Amplitdue')
%     title('Product of Signal and Filter')
%     subplot(3,1,3), plot(ks,abs(fftshift(vgt)/max(abs(fftshift(vgt)))))
%     xlabel('Time(sec)'); ylabel('Amplitude')
%     title('FFT of Product')
%     axis([-50 50 0 0.3])
%     drawnow
%     pause(0.1)
end
figure(3)
subplot(2,2,1)
pcolor(tslide,ks,vgt_spec.'), shading interp
xlabel('Time(sec)'); ylabel('Frequency (\omega)')
title('a = 500')
colormap(hot)


%-----------------------------------------------------------------------

vgt_spec=[];
a = 250;
tslide=0:dt:t(end-1);
```

```
for j=1:length(tslide)
    g=exp(-a*(t-tslide(j)).^2); % Gabor
    vg=g.*v; vgt=fft(vg);
    vgt_spec=[vgt_spec; abs(fftshift(vgt))];

end

subplot(2,2,2)
pcolor(tslide,ks,vgt_spec.'), shading interp
xlabel('Time(sec)'); ylabel('Frequency (\omega)')
title('a = 250')
colormap(hot)

%-----------------------------------------------------------------

vgt_spec=[];
a = 50;
tslide=0:dt:t(end-1);
for j=1:length(tslide)
    g=exp(-a*(t-tslide(j)).^2); % Gabor
    vg=g.*v; vgt=fft(vg);
    vgt_spec=[vgt_spec; abs(fftshift(vgt))];

end

subplot(2,2,3)
pcolor(tslide,ks,vgt_spec.'), shading interp
xlabel('Time(sec)'); ylabel('Frequency (\omega)')
title('a = 50')
colormap(hot)

%-----------------------------------------------------------------


vgt_spec=[];
a = 2;
tslide=0:dt:t(end-1);
for j=1:length(tslide)
    g=exp(-a*(t-tslide(j)).^2); % Gabor filter Gaussian
    vg=g.*v; vgt=fft(vg);
    vgt_spec=[vgt_spec; abs(fftshift(vgt))];

end

subplot(2,2,4)
pcolor(tslide,ks,vgt_spec.'), shading interp
xlabel('Time(sec)'); ylabel('Frequency (\omega)')
title('a = 2')
colormap(hot)

%-----------------------------------------------------------------


% Undersampling

figure(4)
a = 250;
```

```
dt = 0.5;
vgt_spec=[];
tslide=0:dt:t(end-1);
for j=1:length(tslide)
    g=exp(-a*(t-tslide(j)).^2); % Gabor filter Gaussian
    vg=g.*v; vgt=fft(vg);
    vgt_spec=[vgt_spec; abs(fftshift(vgt))];

end

subplot(2,1,1)
pcolor(tslide,ks,vgt_spec.'), shading interp
xlabel('Time(sec)'); ylabel('Frequency (\omega)')
title('Undersampled Signal, a = 250, dt = 0.5')
colormap(hot)

% Oversampling
a = 50;
dt = 0.01;
vgt_spec=[];
tslide=0:dt:t(end-1);
for j=1:length(tslide)
    g=exp(-a*(t-tslide(j)).^2); % Gabor filter
    vg=g.*v; vgt=fft(vg);
    vgt_spec=[vgt_spec; abs(fftshift(vgt))];

end

subplot(2,1,2)
pcolor(tslide,ks,vgt_spec.'), shading interp
xlabel('Time(sec)'); ylabel('Frequency (\omega)')
title('Oversampled Signal, a = 250, dt = 0.01')
colormap(hot)
%------------------------------------------------------------------


% Mexican hat wavelet


vgt_spec=[];
a = 0.05;

% function for mexican hat wavelet

m = @(x) (1-(x/a).^2).*exp(-(x/((sqrt(2)*a))).^2).*(1/sqrt(a));
dt = 0.1;
tslide=0:dt:t(end-1);
for j=1:length(tslide)
    x = t-tslide(j);
    g=m(x); % Mexican hat
    vg=g.*v; vgt=fft(vg);
    vgt_spec=[vgt_spec; abs(fftshift(vgt))];

end

figure(5)
y = t-tslide(50);
```

```
g = m(y);
subplot(2,1,1)
plot(t,v,'k',t,g,'r')
xlabel('Time(sec)'); ylabel('Amplitude')
axis([4 6 -3 6])
title('Signal with Mexican hat wavelet')

subplot(2,1,2)
pcolor(tslide,ks,vgt_spec.'), shading interp
xlabel('Time(sec)'); ylabel('Frequency (\omega)')
title('Mexican hat wavelet Spectogram, a =0.05, dt = 0.1')
colormap(hot)

%-----------------------------------------------------------------------

% Shannon window
a = 3;

%function for Shannon window

s = @(x) (a*abs(x) <= 0.5);

vgt_spec = [];
dt = 0.1;
tslide=0:dt:t(end-1);
for j=1:length(tslide)
    y = t-tslide(j);
    g=s(y);
    vg=g.*v; vgt=fft(vg);
    vgt_spec=[vgt_spec; abs(fftshift(vgt))];

end

figure(6)
y = t-tslide(40);
g = s(y);
subplot(2,1,1)
plot(t,v,'k',t,g,'r')
xlabel('Time(sec)'); ylabel('Amplitude')
set(gca, 'Fontsize', [15]),
xlim([0, t(end)])
title('Signal with Shannon filter')

subplot(2,1,2)
pcolor(tslide,ks,vgt_spec.'), shading interp

xlabel('Time(sec)'); ylabel('Frequency (\omega)')
title('Shanon Window Spectogram, a =3, dt = 0.1')
set(gca, 'Fontsize', [15]),
colormap(hot)

% Part 2
%-----------------------------------------------------------------------

% Piano

tr_piano=16; % record time in seconds
```

```
y= audioread('music1.wav');
Fs=length(y)/tr_piano; % sampling rate
figure(1)
subplot(2,1,1)
plot((1:length(y))/Fs,y);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Mary had a little lamb (piano) signal');



y = y'/2;
n = length(y);  % length of signal
L = tr_piano;    % final time
t = linspace(0,L, n); % discretized time vector
k = (2*pi/L)*[0:n/2-1 -n/2:-1];  %rescale
ks = fftshift(k); %shift

yt = fft(y);  %Fourier transform

subplot(2,1,2)
plot(ks, abs(fftshift(yt))/max(abs(yt)));
xlabel('Frequency'), ylabel('FFT(y)')
title('FFT of signal')

%Gabor filtering

a = 500; % width parameter
dt = 0.25;
ygt_spec = [];
tslide = 0:dt:t(end-1);
for j=1:length(tslide)
    g=exp(-a*(t-tslide(j)).^2); % Gabor
    yg=g.*y; ygt=fft(yg);
    ygt_spec=[ygt_spec; abs(fftshift(ygt))];

end

figure(2)
pcolor(tslide,ks/(2*pi),(ygt_spec).'), shading interp
xlabel('Time(sec)'); ylabel('Frequency (Hz)')
title('Spectogram (Piano), a =500, dt = 0.1')
set(gca, 'Fontsize', [15]),
ylim([0, 950])
colormap(hot)

%-----------------------------------------------------------------------

% Recorder

tr_rec=14; % record time in seconds
y=audioread('music2.wav');
Fs=length(y)/tr_rec;
figure(3)
subplot(2,1,1)
plot((1:length(y))/Fs,y);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Mary had a little lamb (recorder) signal');
```

```
y = y'/2;
n = length(y);  % length of signal
L = tr_rec;    % final time
t = linspace(0,L, n); % discretized time vector
k = (2*pi/L)*[0:n/2-1 -n/2:-1];  %rescale
ks = fftshift(k); %shift

yt = fft(y);  %Fourier transform

subplot(2,1,2)
plot(ks, abs(fftshift(yt))/max(abs(yt)));
xlabel('Frequency'), ylabel('FFT(y)')
title('FFT of signal')

%Gabor filtering

a = 500; % width parameter
dt = 0.1;
ygt_spec = [];
tslide = 0:dt:t(end-1);
for j=1:length(tslide)
    g=exp(-a*(t-tslide(j)).^2); % Gabor
    yg=g.*y; ygt=fft(yg);
    ygt_spec=[ygt_spec; abs(fftshift(ygt))];

end

figure(4)
pcolor(tslide,ks/(2*pi),(ygt_spec).'), shading interp
xlabel('Time(sec)'); ylabel('Frequency (Hz)')
title('Spectogram (Recorder), a =500, dt = 0.1')
set(gca, 'Fontsize', [15]),
ylim([0, 2000])
colormap(hot)
```

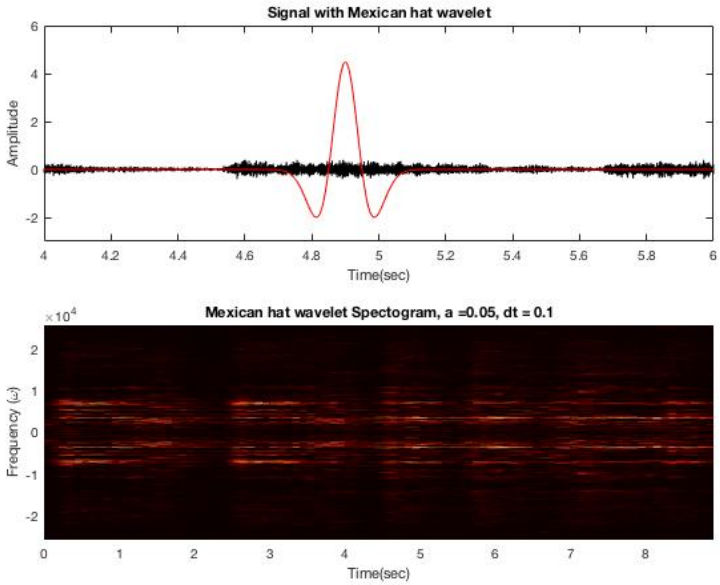

Figure 5: The top panel contains the Mexican hat filter for $a = 0.05$ and $dt = 0.1$. The bottom panel contains the spectogram produced.

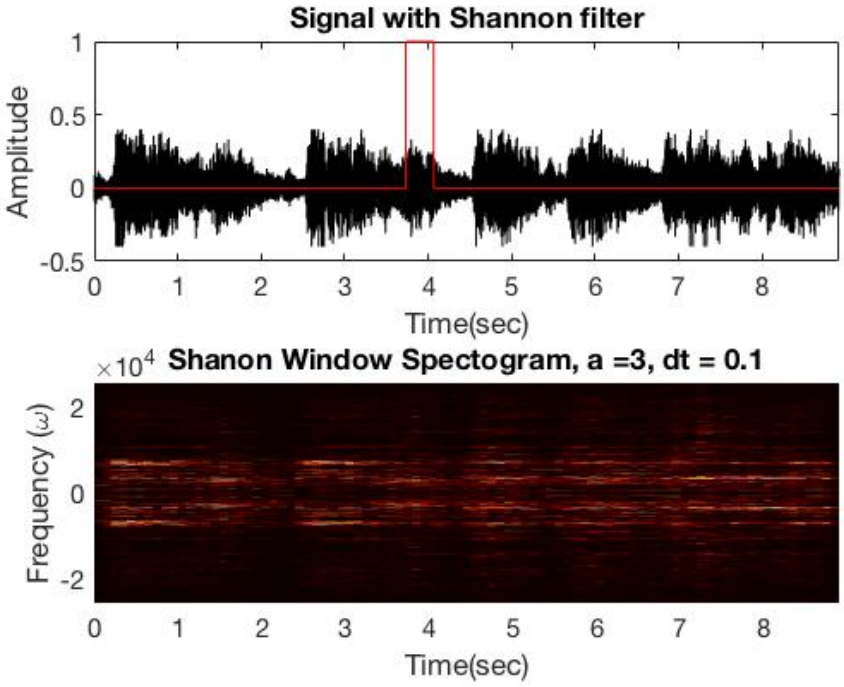

Figure 6: The top panel contains the Shannon filter for $a = 3$ and $dt = 0.1$. The bottom panel contains the spectogram produced.

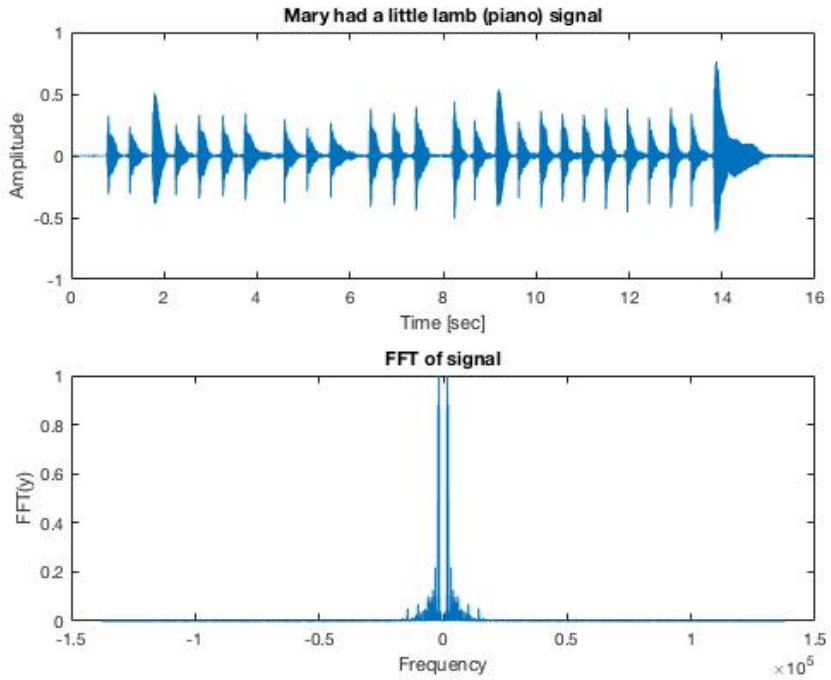

Figure 7: The top panel contains the signal from a piano. The bottom panel contains the Fourier Transform of the signal.

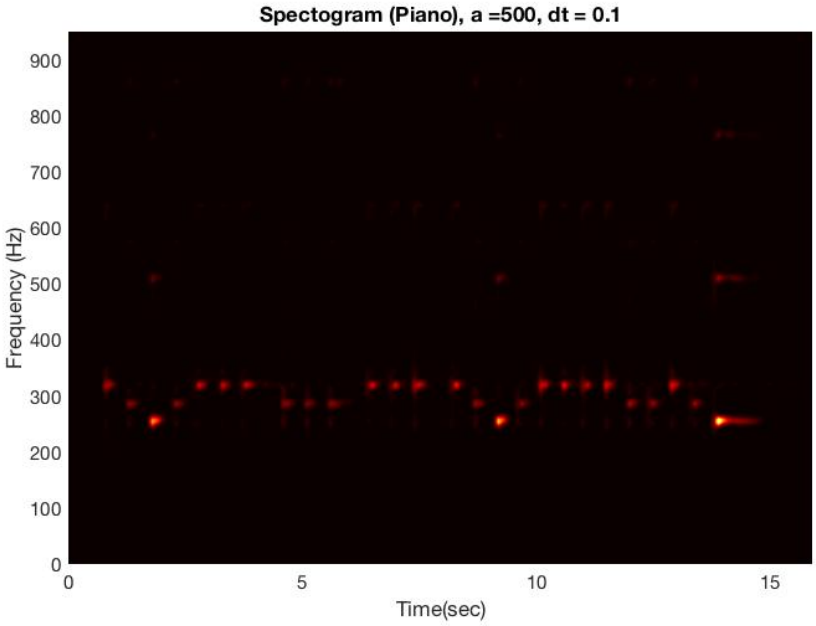

Figure 8: Spectogram of the piano using Gaussian window.

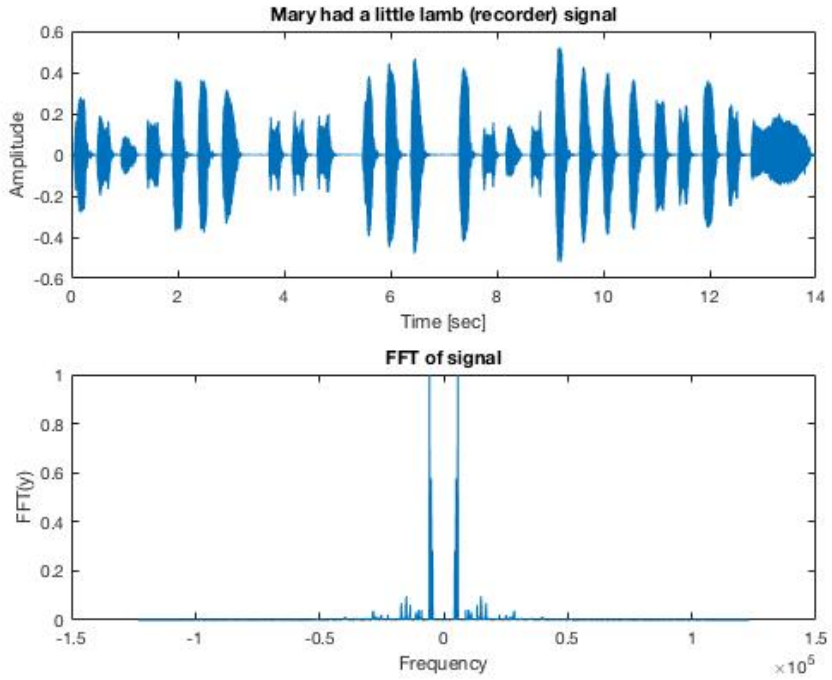

Figure 9: The top panel contains the signal from a recorder. The bottom panel contains the Fourier Transform of the signal.

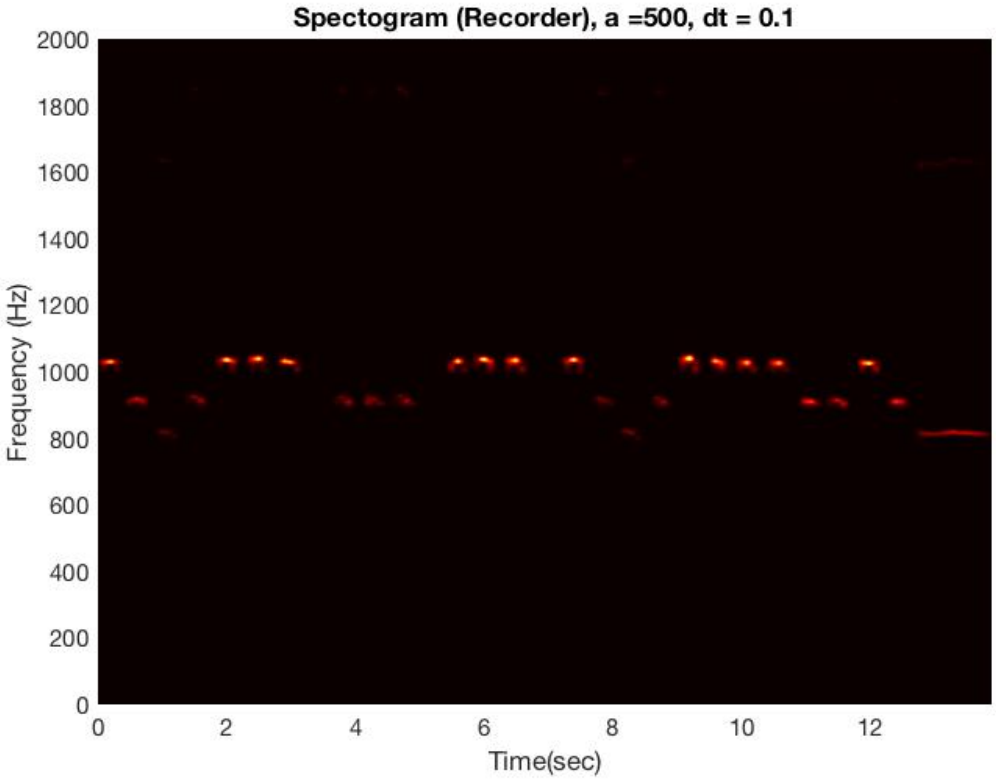

Figure 10: Spectogram of the recorder using Gaussian window.