Amath 582 Homework 2

Due: January 28, 2016

Brian de Silva[1]

Abstract
We accomplish two major tasks in this assignment. First we perform numerical experiments to uncover relationships between different Gabor transforms, their associated parameters, and the spectrograms they produce. We then apply a particular Gabor transform to two recordings of the song *Mary had a little lamb* on different instruments and reconstruct their musical scores using spectrograms.

[1] *Department of Applied Mathematics, University of Washington, Seattle*

Contents

1. Introduction

This assignment deals with time-frequency analysis and has two parts. In Part I we analyze the frequencies present at different times in a segment of Handel's Messiah. We use this as an opportunity to experiment with various Gabor windows and the parameters associated with them such as window width and time step length. We explore the spectrograms generated by the various parameter choices. We then provide some analysis of our findings.

In Part II we are given two recordings of the hit song *Mary had a little lamb*, one played on the piano and the other on the recorder. We use our newfound expertise to reconstruct the music score for each audio file using just their spectrograms (and a table of the frequencies corresponding to each musical note). We then compare and contrast the spectrograms of the song performed on the two instruments.

This writeup is structured as follows: in Section 2 we give a brief review of the theory behind our techniques. Next we discuss our algorithms and their implementations in MATLAB in Section 3 before discussing our numerical results in Section 4. Finally we make some closing remarks in Section 5.

2. Theoretical Background

2.1 Gabor transforms

The Fast Fourier Transform (FFT) is an indispensible tool for analyzing the frequency content of a given signal However it has at least one major drawback, it identifies all the frequencies present in a data set simultaneously, without giving any information about *when* they occured, i.e. when taking an FFT we lose all temporal information (or spatial information if we interpret the independent variable as being space-like). Gabor transforms are an attempt to remedy this issue. The novel idea behind Gabor transforms is simply to make a slight adjustment to the kernel used in Fourier Transforms so that only local (in time) information is transformed into the frequency domain. By varying a parameter in the transform one can shift around the instant in time from which one wishes to extract frequency information. By varying another one can specify how wide a range of times is considered "local".

The Fourier Transform is given by

$$\int_{-\infty}^{\infty} f(\tau)e^{-ik\tau}d\tau = (f, e^{ikt}). \tag{1}$$

Note that it may be expressed as an inner product between f and $e^{ik\tau}$. Given the kernel

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

we define the *Gabor transform* of an L^1 function f as the inner product between f and $g_{t,k}$:

$$\tilde{f}_g(t,k) = \int_{-\infty}^{\infty} f(\tau)\bar{g}(\tau - t)e^{-ik\tau}d\tau = (f, g_{t,k}). \tag{3}$$

The function $g(\tau - t)$ is some function which is typically very small outside some radius of 0, a, of 0. Its purpose is to create a window of time outside of which the values of f are severely damped, and inside of which its output is relatively unchanged. For example, in this assignment we experiment with the following window functions:

$$g(t) = e^{-at^2} \qquad \text{(Gaussian)} \tag{4}$$

$$g(t) = e^{-at^{10}} \qquad \text{(Super-Gaussian)} \tag{5}$$

$$g(t) = \left(1 - \left(\frac{t}{a}\right)^2\right) e^{-\left(\frac{t}{\sqrt{2}a}\right)^2} \qquad \text{(Mexican Hat)} \tag{6}$$

$$g(t) = \begin{cases} 0 & |t| > 1/2 \\ 1 & |t| \le 1/2 \end{cases} \qquad \text{(Step function).} \tag{7}$$

There is a tradeoff which is in some sense inherent in using Gabor transforms. We are able to retain some temporal information when moving into the frequency domain, but the range of frequencies which we can resolve decreases. This is because using a window which does not encompass the entire temporal domain means that very long frequencies present in the signal will not be captured within the window and therefore will not appear in the spectrogram. Taking a window that is large enough to resolve these frequencies means sacrificing temporal precision and aiming for extreme temporal resolution by taking a very small window results in the loss of the ability to detect all but the shortest frequencies. These tradeoffs are explored more fully in Section 3.1.

Throughout this section we have mentioned spectrograms without having defined them. To construct one we first discretize the temporal domain into suitably small subintervals by selecting equally spaced points in time. For each of these points we apply a Gabor filter centered at the point to the original signal, take a FFT of the filtered signal, and stack the resulting frequency profiles in a 2-D array. We then use the MATLAB function `pcolor` to plot the absolute value of the resulting array. This plot is the spectrogram. It gives one an idea of which frequencies are prominent in the signal at each of the selected points in time. Some examples are shown in 4.

3. Algorithms Implementation and Development

In this section we describe the numerical algorithms developed to explore various parameters associated with Gabor transforms and to determine the scores of the piano and recorder sound clips. The procedures are given below along with a few comments and pertinent details.

3.1 Part I

Below we outline the MATLAB script written to allow experimentation with over and undersampling, various Gabor windows, and Gabor window widths.

1. Read in the audio file and take its DFT

This gives us an idea of the frequency components present in the unmodified data. Note that all temporal information is lost once the FFT is taken. As before we must scale the wavenumbers by $2\pi/L$ since MATLAB's `fft2` function assumes its input to come from a 2π-periodic function. Here L is the length of the interval of interest. Since the audio file produces a vector with an odd number of entries, in order to produce the right number of frequency components, we identify the first and last entries of the vector. The FFT assumes the data to be 2π-periodic so our actions are somewhat justified. It turns out that the first and last entries are both zero in any case.

2. Create the desired Gabor window

As detailed in Section 2, we test out a standard Gaussian filter, a super-Gaussian, the Mexican Hat wavelet, and a step function. At this stage we also specify the width of the window as well as the time discretization spacing, dt.

3. Create a spectrogram by evolving time in increments of dt

At each time step we apply the window chosen in step 2 to the data, take the FFT of the filtered data, then store the magnitude of the result in a row of the 2-dimensional spectogram array.

4. Plot the spectrogram

We use the `pcolor` function in MATLAB with the `shading interp` option and the hot colormap to plot the spectrogram.

3.2 Part II

Below we sketch the alrogithm used to reconstruct the music score for the two audio clips of *Mary had a little lamb* and to compare the spectrograms of the two.

1. Read in the audio files and take their DFTs

This step mimicks that of the first step in Section 3.1 except the sound vectors are both even in length so no modification is necessary.

2. Create the Gabor window

Here we use the standard Gaussian Gabor window introduced in lecture which is given explicity in Section 2. We also specify the width of the window and the time discretization spacing dt. We had success using the same parameters for both of the two sound files in Part II.

3. Create spectrograms by evolving time in increments of dt

This step is the same as in Part I.

4. Plot the spectrograms

We use the same options as in Part I to construct the spectrogram plots.

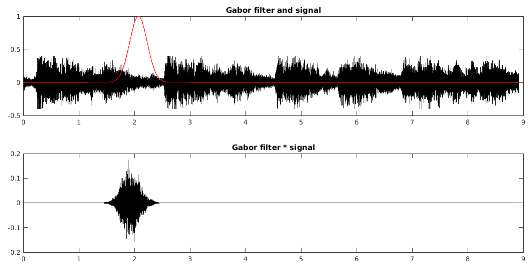


Figure 1. Top: The Guassian filter along with the Handel signal, Bottom: The signal after the filter has been applied (width=20)

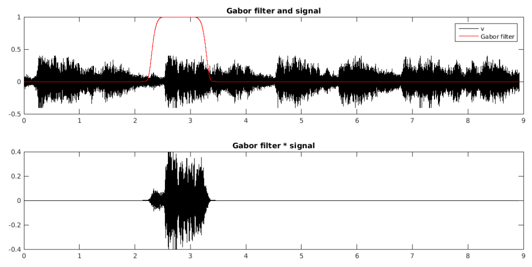


Figure 2. Top: The super-Guassian filter along with the Handel signal, Bottom: The signal after the filter has been applied (width=1000)

4. Computational Results

In this section we first outline the findings of our numerical experiments from testing various combinations of parameters in the algorithm in Section 3.1. We then recover the musical scores of the recorder and piano audio files and then compare and contrast the frequency signatures of the two instruments.

4.1 Part I

For each of the different choices of windows we experimented with various window lengths, ranging from very narrow to large. Plots of the four filters are shown in Figures 1 - 4 and the original Handel signal and its FFT are shown in Figure 5.

We started with the Gaussian filter used in lecture. Many of the relationships between parameters and the resulting spectrograms will hold for the other types of filters. We expect narrower windows to grant us better temporal resolution at the cost of limiting the range of frequencies represented in the spectrogram. In particular we miss some of the low frequency behavior of the original signal if the window is narrower than the associated wavelengths of the low frequencies. For larger windows we better capture the lower wavenumber components but we lose temporal precision. Choosing a window size that is somewhere in the middle should give us some accuracy in the time domain as well as the ability to represent some of the lower frequencies.

These properties are evident in Figures 6 - 8 which show the spectrograms generated by Gaussian filters of short, medium, and long lengths. Notice how in Figure 6 the frequency com-

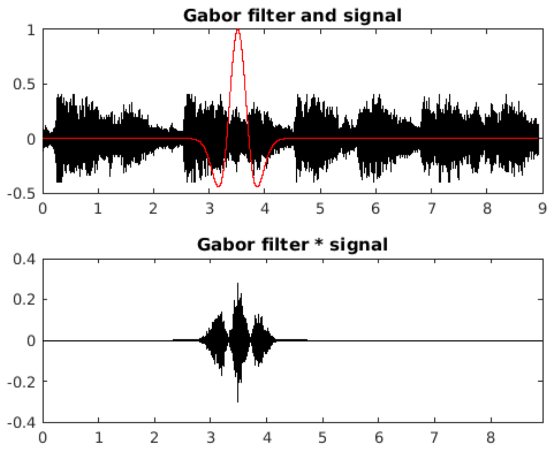


Figure 3. Top: The Mexican Hat filter along with the Handel signal, Bottom: The signal after the filter has been applied (width=0.2)

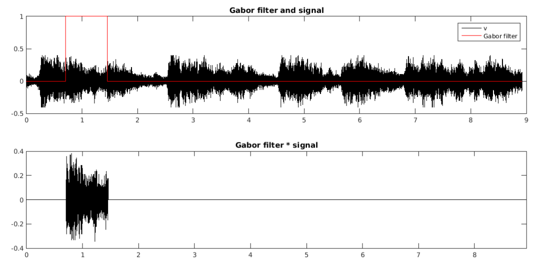


Figure 4. Top: The Step function filter along with the Handel signal, Bottom: The signal after the filter has been applied (width=0.75)

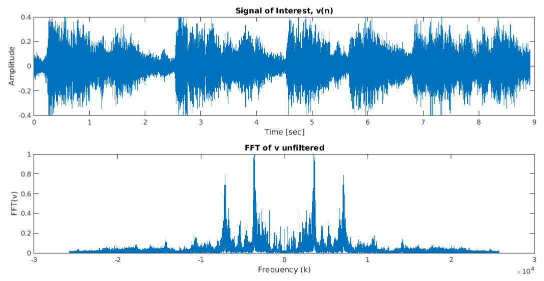


Figure 5. A snippet of Handel's Messiah and its FFT

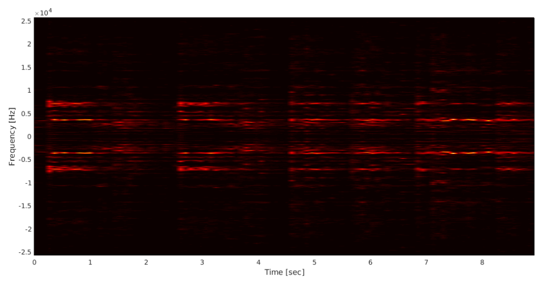


Figure 6. Spectrogram of Handel signal using a narrow Gaussian filter (width=2000)

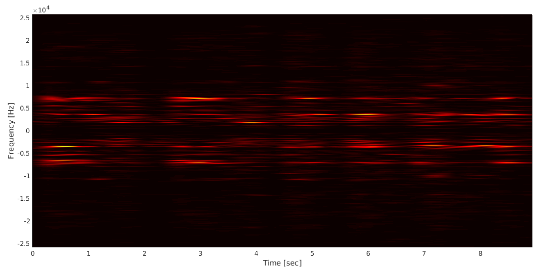


Figure 7. Spectrogram of Handel signal using a Gaussian filter (width=20)

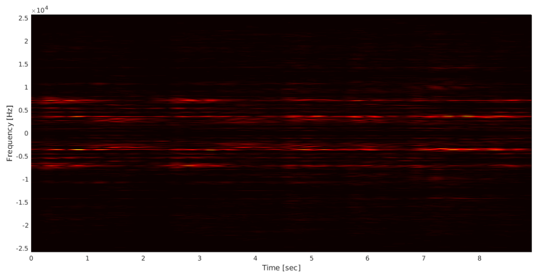


Figure 10. Spectrogram of Handel signal using a Mexican Hat filter (width=0.2)

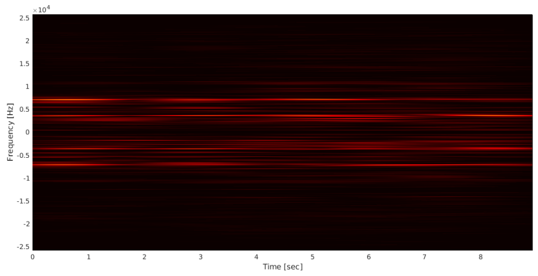


Figure 8. Spectrogram of Handel signal using a wide Gaussian filter (width=1)

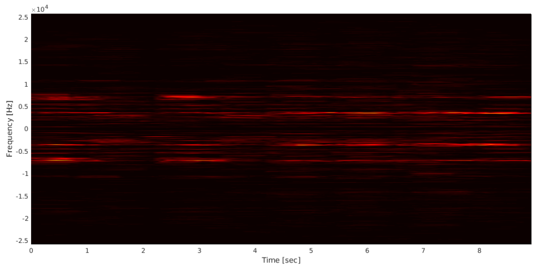


Figure 11. Spectrogram of Handel signal using a step function filter (width=0.75)

ponents arise and vanish in sharp jumps. Contrast this with Figure 8 where the componenets are stretched out horizontally. This occurs because it takes many time steps for a frequency to enter or leave the filter as it slides along. If you look closely, however, there are more frequency components present near the (vertical) center of the spectrogram in Figure 8 than in Figure 6 as we predicted. The medium-sized filter produces a spectrogram somewhere between the other two. We used 100 time steps in constructing each of these spectrograms.

Since varying the width of the other filters changes the resulting spectrograms in the same manner as with the Gaussian filter, and for the sake of brevity, we omit further discussion of the effect of window size on spectrograms. Instead we now focus on the qualitative differences between the spectrograms produced by different filters.

Figures 9 - 11 show the spectrograms produced by the super-Gaussian, Mexican Hat, and step function windows of comparable size to that of the Gaussian filter previously used. The super-Gaussian spectrogram looks essentially like a slightly more washed out version of its Gaussian counterpart. This may be because the super-Gaussian filter is close to 1 in value for a larger range of inputs causing frequencies to persist longer during its sweep, whereas the Gaussian filter drops off away from its center causing more localization (in time). The Mexican Hat filter appears to do a better job at capturing longer frequencies than either of the previous two, probably because it does not decay to zero as quickly as they do. Observe also that there are many “holes” in its spectrogram. These may be caused by the fact that the function both takes on negative values and is zero at two points in the domain at any given time. The step function filter actually produces a spectrogram which looks remarkably like that of the super-Gaussian, but perhaps we should expect this since the two filters closely resemble one another in shape.

We also explored the effects of sampling rate on the appearence of the spectrogram. Figure 12 displays a spectrogram

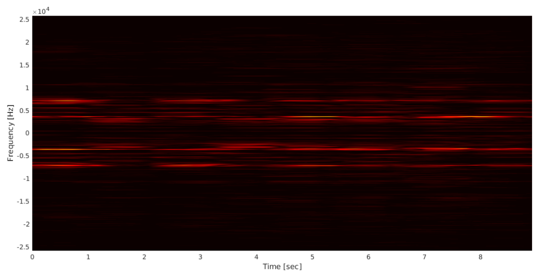


Figure 9. Spectrogram of Handel signal using a super-Gaussian filter (width=1000)

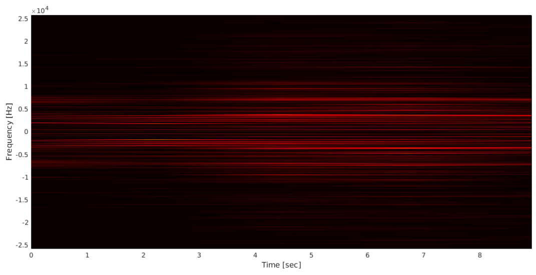


Figure 12. Spectrogram of undersampled Handel signal using Gaussian window (width=20, 5 time steps)

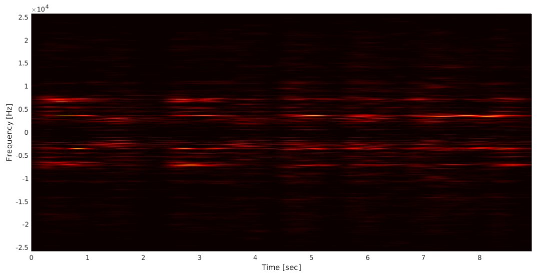


Figure 13. Spectrogram of oversampled Handel signal using Gaussian window (width=20, 1000 time steps)

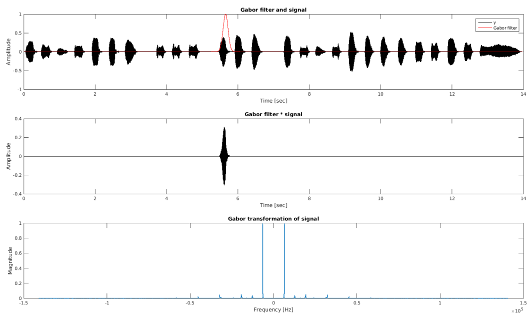


Figure 14. Top: plot of sound clip of Mary had a little lamb played on the recorder and the Gabor filter at one instant in time. Middle: Gabor filtered signal. Bottom: Magnitude of the Gabor transform of the sound data corresponding to the above filter.

which was created using only five time steps (and a Gaussian filter). The frequencies are extremely washed out (partly because of the `shading interp` command being used). The spectrogram picks up a few frequencies at each time step, but it is not at all clear when each arises or disappears from the original signal. Figure 13, on the other hand, gives a spectrogram obtained using 1000 time steps and the same filter. Compare this with Figure 7 where only 100 time steps were used. The two look nearly identical, yet the oversampled spectrogram took much longer to create and even longer to plot. Taken together these results suggest that increasing the sampling rate can improve temporal accuracy at first, but it eventually provides diminishing returns.

4.2 Part II

Figures 14 and 15 show the record/piano audio signals, the Gabor filters used, and the Gabor transforms for particular choices of the parameter t.

Using the methods outlined in Section 3.2 we produce the recorder spectrograms shown in Figures 16 and 17. The second just shows a subset of the first in a region of interest. This region of interest is simply the frequency components of largest magnitude. Since the plot is colored using a logarithmic scale the brighter spots actually correspond to components of much much larger magnitude. The dimmer spots only occur at multiples of the principal frequencies observed

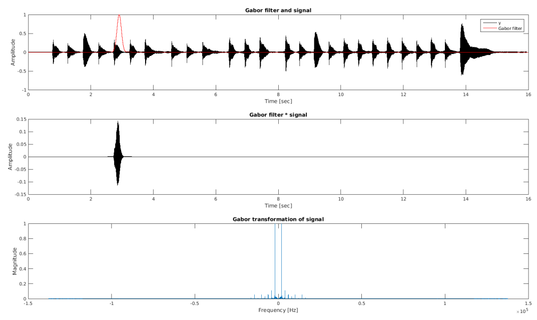


Figure 15. Top: plot of sound clip of Mary had a little lamb played on the piano and the Gabor filter at one instant in time. Middle: Gabor filtered signal. Bottom: Magnitude of the Gabor transform of the sound data corresponding to the above filter.

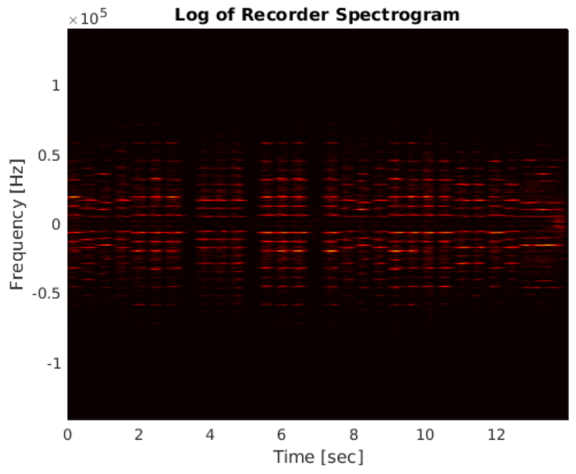


Figure 16. A plot of the log of the spectrogram for the recorder signal (100 timesteps, width = 100).

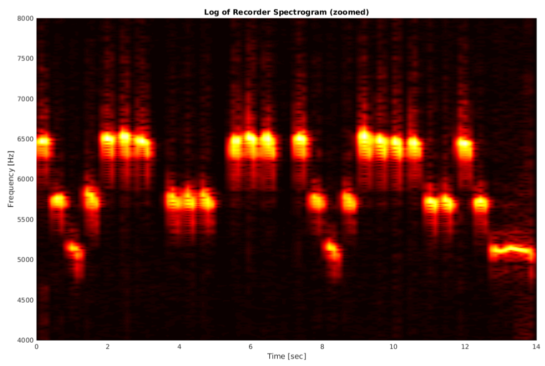


Figure 17. A zoomed in plot of the log of the spectrogram for the recorder signal (100 timesteps, width = 100).

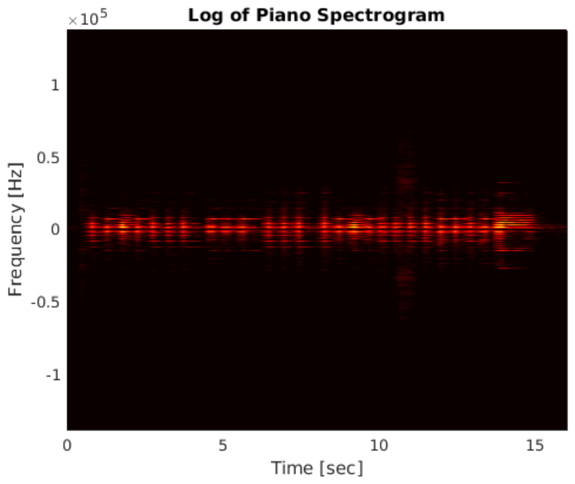


Figure 18. A plot of the log of the spectrogram for the piano signal (100 timesteps, width = 100).

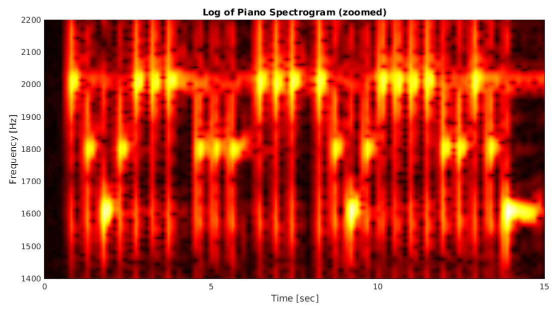


Figure 19. A zoomed in plot of the log of the spectrogram for the piano signal (100 timesteps, width = 100).

in Figure 16. This is the result of the timbre of the instrument. We can essentially just read off the notes present from the magnitudes of the frequency components. They correspond to the following notes (from left to right): **B A G A B B B A A B B B B A G A B B B B A A B A G**.

Figures 18 and 19 show similar spectrograms for the piano. The notes are the same, but are a few octaves lower. Notice how much more noise there seems to be near the principal frequencies. This is related to the fact that the piano generates more overtones than the recorder–its notes are fuller and richer than those of the recorder. Once we zoom in on the center frequencies it is easy to tell the two apart.

5. Summary and Conclusions

In Part I of this assignment we confirmed our intuitions that as we moved from narrow to wide window sizes our ability to temporally resolve frequencies decreased, but we slowly gained the ability to increasingly lower frequencies. We saw that undersampling and oversampling had somewhat similar effects on the spectrograms as using narrow and wide windows, respectively. We also observed some differences between different choices of filers.

In Part II of this assignment we successfully used Gabor transforms and spectrograms to reconstruct the musical scores of two renditions of *Mary had a little lamb*.

Appendix A: MATLAB Functions

Here we outline the nonstandard MATLAB functions used to complete this assignment.

`audioread('filename')`: This function reads the sound file specified by `filename` and returns two outputs: a vector of sampled sound data and the sample rate for said data.

`audioplayer(y,Fs)`: This command creates an audioplayer object which can play the sound stored in the vector `y` with sampling rate `Fs`. It is typically paired with the `playblocking` function which actually plays the sound.

`fft(X)`: Given a $1-$dimensional array of data, `X`, `fft2(X)` returns the $1-$dimensional DFT of `X`. The output has the same size as the input. To plot the frequency components given by `fft` one should first apply the `fftshift` command as `fft` returns a shifted version of the frequency components.

`fftshift(v)`: This command shifts the vector/matrix output of `fft`, `fft2`, and `fftn` so that the 0 frequency lies at the center of the vector/matrix. For example, in one dimension `v = fft(data)` is a vector and `fftshift(v)` swaps the first and second halves of `v`. This command is useful for plotting the FFT of data.

`pcolor(X,Y,vals)`: Plots the scalars stored in the 2-D array `vals` at the points specified by the 2-D arrays `X` and `Y` by mapping them to colors. `X`, `Y`, and `vals` should have the same dimensions. `X` and `Y` can be created with the `meshgrid` command.

`playblocking(AP)`: `playblocking` takes an audioplayer object, `AP` as an input and plays the associated sound clip.

Appendix B: MATLAB Code

See the following pages published in MATLAB for the implementation of the algorithm presented in Section 3.

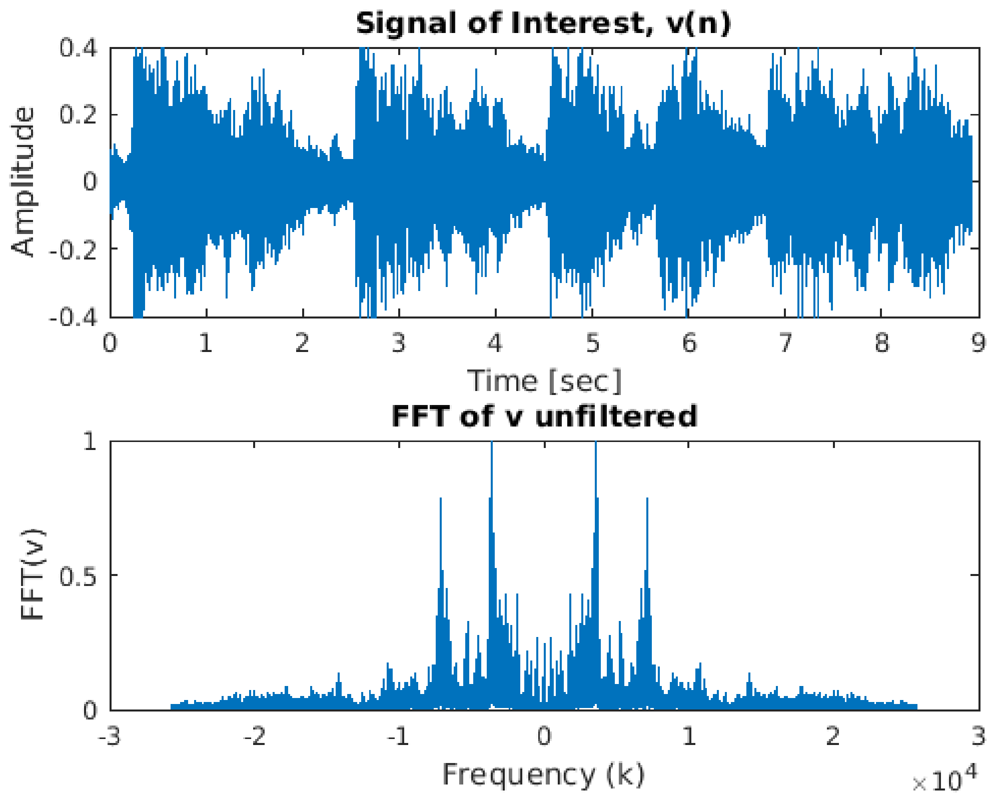
Table of Contents

```
% Amath 582 HW 2
% Brian de Silva
% 1422824

clear all; close all;
```

Part 1

```
load handel
v = y'/2;
subplot(2,1,1)
plot((1:length(v))/Fs,v);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Signal of Interest, v(n)');

% % Play back audio
% p8 = audioplayer(v,Fs);
% playblocking(p8);

% Identify first and last points of v
v = v(1:end-1);

L = (length(v)-1) / Fs;
t1 = (0:length(v))/Fs;
t = t1(1:end-1);
n = length(v);
k = (2*pi/L)*[0:n/2 - 1, -n/2:-1];
ks = fftshift(k);

% Look at frequency components of entire signal
vt = fft(v);
subplot(2,1,2)
plot(ks, abs(fftshift(vt)) / max(abs(vt)));
xlabel('Frequency (k)')
ylabel('FFT(v)')
title('FFT of v unfiltered')


% Create Gabor filter for spectrogram
width = 0.75;                                                      %
 Width of filter
```

```
numsteps = 1000;                                                %
 Number of time steps to take
tslide = linspace(0,t(end-1),numsteps);             % Time
 discretization
spec = zeros(length(tslide),length(v));                % Preallocate
 space for spectrogram

% Create multiple Gabor filters for testing
% { Gaussian, super Gaussian, Mexican Hat, step function }
filter = {@(x) exp(-width*(x).^2), @(x) exp(-width*(x).^10),@(x)
 (1-(x/width).^2).*exp(-((x/width).^2)/2), @(x) (x>-width/2 & x<
 width/2)};


% figure()
for j=1:length(tslide)
    g = filter{4}(t-tslide(j));                        % Gabor filter
 (choose 1-4 for four different filters)
    vg = g.*v;                                            % Apply Gabor
 filter
    vgt = fft(vg);                                       % Take fft of
 filtered data
    spec(j,:) = abs(fftshift(vgt));                % Store fft in
 spectrogram

    % Make some cool movies of what the Gabor transform is
 accomplishing
%     subplot(2,1,1), plot(t,v,'k',t,g,'r'), title('Gabor filter and
 signal'), legend('v','Gabor filter')
%     subplot(2,1,2), plot(t,vg,'k'), title('Gabor filter * signal')
%     axis([0 t(end) -0.4 0.4])
% %     subplot(3,1,3), plot(ks, abs(fftshift(vgt))/max(abs(vgt))),
 title('Gabor transformation of signal')
%     drawnow

end

% Plot spectrogram
% figure()
% pcolor(tslide,ks,spec.'), shading interp
%  colormap('hot'), xlabel('Time [sec]'), ylabel('Frequency [Hz]')
```



Part II - Mary had a little lamb

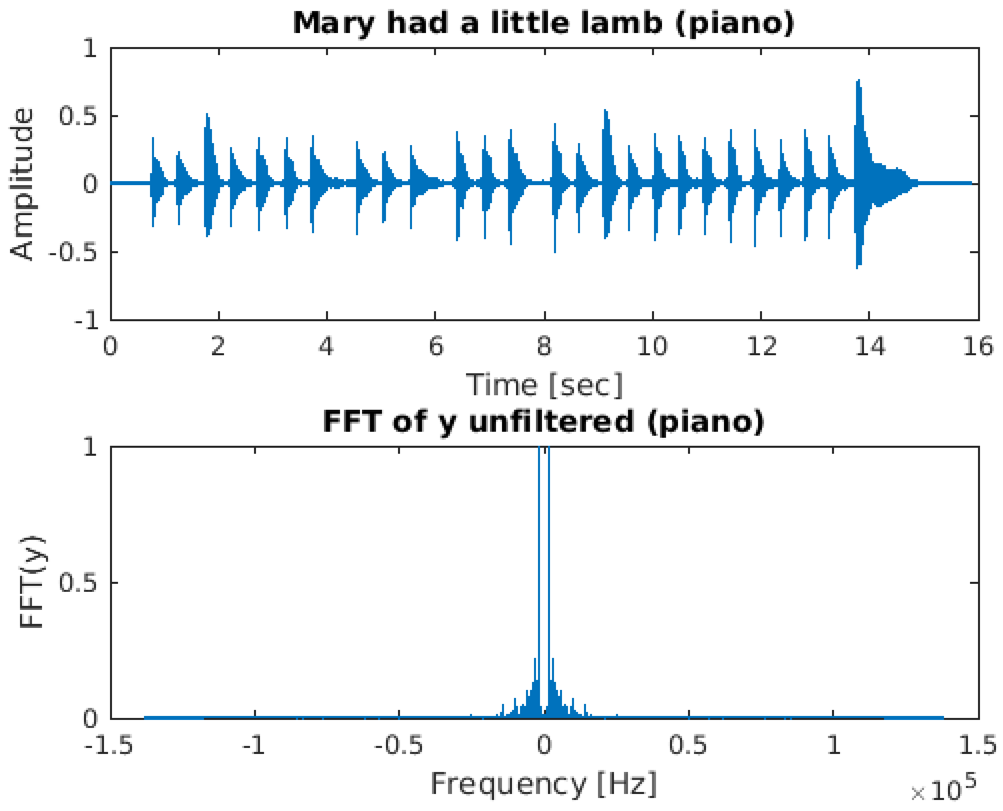
```
clear all;
```

Piano

music1.wav plays mhall on piano

```
tr_piano=16;                          % Record time in seconds
% y=wavread('music1').';      % Read music
[y,Fs] = audioread('music1.wav');
y = y.';
% Fs=length(y)/tr_piano;
figure()
subplot(2,1,1)
plot((1:length(y))/Fs,y);        % Plot data
xlabel('Time [sec]'); ylabel('Amplitude');
title('Mary had a little lamb (piano)'); drawnow

% Play music
% p8 = audioplayer(y,Fs);
% playblocking(p8);

L = tr_piano;
t1 = linspace(0,16,length(y)+1);
```

```
t = t1(1:end-1);
n = length(y);
k = (2*pi/L)*[0:n/2 - 1, -n/2:-1];
ks = fftshift(k);

% Look at frequency components of entire signal
yt = fft(y);
subplot(2,1,2)
plot(ks, abs(fftshift(yt)) / max(abs(yt)));
xlabel('Frequency [Hz]'), ylabel('FFT(y)'), title('FFT of y unfiltered
 (piano)')

% Create Gabor filter for spectrogram
width = 100;                                                      %
 Width of filter
numsteps = 100;                                                 % Number
 of time steps to take
tslide = linspace(0,t(end-1),numsteps);             % Time
 discretization
spec = zeros(length(tslide),length(y));                % Preallocate
 space for spectrogram

% Create spectrogram using Gabor filter
% figure()
for j=1:length(tslide)
    g = exp(-width*(t - tslide(j)).^2);     % Gabor filter
    yg = g.*y;
    ygt = fft(yg);
    spec(j,:) = abs(fftshift(ygt));             % Store fft in
 spectrogram

    % Make some cool movies of what the Gabor transform is
 accomplishing
%     subplot(3,1,1), plot(t,y,'k',t,g,'r'), title('Gabor filter and
 signal'), legend('y','Gabor filter')
%     xlabel('Time [sec]'), ylabel('Amplitude')
%     subplot(3,1,2), plot(t,yg,'k'), title('Gabor filter * signal')
%     xlabel('Time [sec]'), ylabel('Amplitude')
%     subplot(3,1,3), plot(ks, abs(fftshift(ygt))/max(abs(ygt))),
 title('Gabor transformation of signal')
%     xlabel('Frequency [Hz]'), ylabel('Magnitude')
%     drawnow
%
end


% Plot relevant portion of spectrogram
% figure()
% pcolor(tslide,ks,log(spec.'+1)), shading interp
% axis([0 15 1400 2200])
% colormap('hot'), xlabel('Time [sec]'), ylabel('Frequency [Hz]'),
 title('Log of Piano Spectrogram (zoomed)')

% Plot full spectrogram
% figure()
% pcolor(tslide,ks,log(spec.'+1)), shading interp
% colormap('hot'), xlabel('Time [sec]'), ylabel('Frequency [Hz]'),
 title('Log of Piano Spectrogram')
```

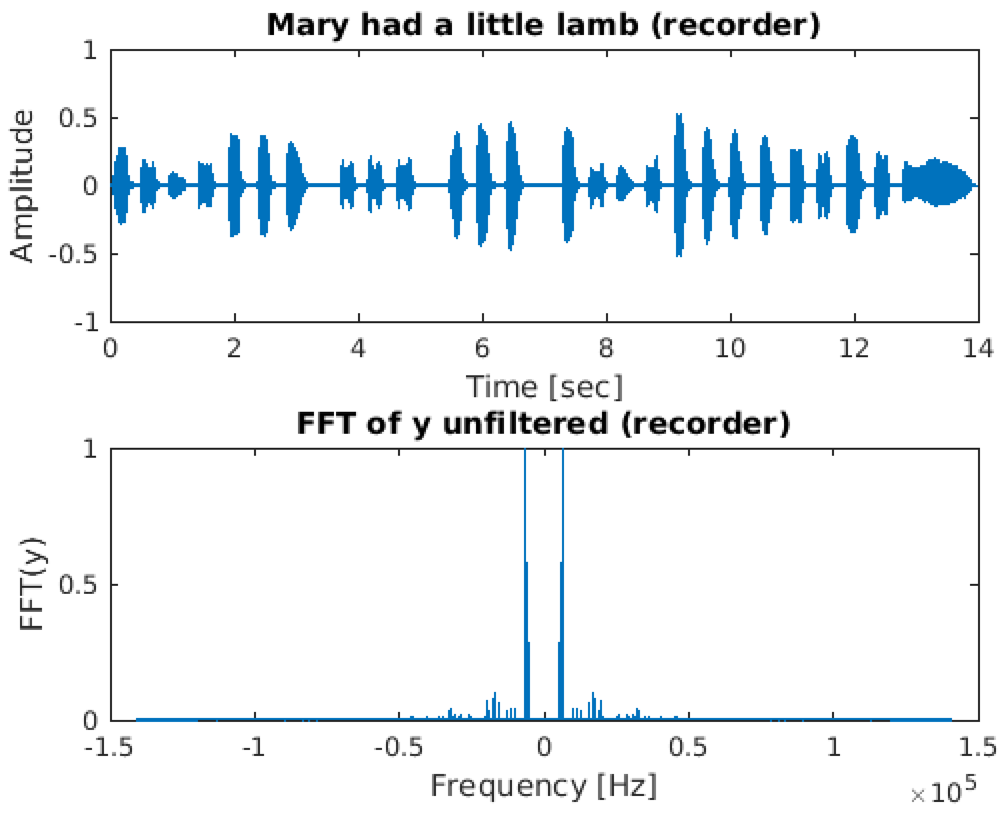


Recorder

music2.wav plays mhall on recorder

```
% figure()
tr_rec=14;                              % Record time in seconds
[y,Fs1]=audioread('music2.wav');
y = y.';
Fs=length(y)/tr_rec;
subplot(2,1,1)
plot((1:length(y))/Fs,y);   % Plot data
xlabel('Time [sec]'); ylabel('Amplitude');
title('Mary had a little lamb (recorder)');

% % Play music
% p8 = audioplayer(y,Fs);
% playblocking(p8);

L = tr_rec;
t1 = linspace(0,14,length(y)+1);
t = t1(1:end-1);
```

```
n = length(y);
k = (2*pi/(L))*[0:n/2 - 1, -n/2:-1];
ks = fftshift(k);

% Look at frequency components of entire signal
yt = fft(y);
subplot(2,1,2)
plot(ks, abs(fftshift(yt)) / max(abs(yt)));
xlabel('Frequency [Hz]'), ylabel('FFT(y)'), title('FFT of y unfiltered
 (recorder)')

% Create Gabor filter for spectrogram
width = 100;                                                   %
 Width of filter
numsteps = 100;                                            % Number
 of time steps to take
tslide = linspace(0,t(end-1),numsteps);          % Time discretization
spec = zeros(length(tslide),length(y));             % Preallocate
 space for spectrogram

% Create spectrogram using Gabor filter
% figure()
for j=1:length(tslide)
    g = exp(-width*(t - tslide(j)).^2);     % Gabor filter
    yg = g.*y;
    ygt = fft(yg);
    spec(j,:) = abs(fftshift(ygt));             % Store fft in
 spectrogram

%     % Make some cool movies of what the Gabor transform is
 accomplishing
%     subplot(3,1,1), plot(t,y,'k',t,g,'r'), title('Gabor filter and
 signal'), legend('y','Gabor filter')
%     xlabel('Time [sec]'), ylabel('Amplitude')
%     subplot(3,1,2), plot(t,yg,'k'), title('Gabor filter * signal')
%     xlabel('Time [sec]'), ylabel('Amplitude')
%     subplot(3,1,3), plot(ks, abs(fftshift(ygt))/max(abs(ygt))),
 title('Gabor transformation of signal')
%     xlabel('Frequency [Hz]'), ylabel('Magnitude')
%     drawnow

end


% % Plot relevant portion of spectrogram
% figure()
% pcolor(tslide,ks,log(spec.'+1)), shading interp
% axis([0 14 4000 8000])
% colormap('hot'), xlabel('Time [sec]'), ylabel('Frequency [Hz]'),
 title('Log of Recorder Spectrogram (zoomed)')

% % Plot full spectrogram
% figure()
% pcolor(tslide,ks,log(spec.'+1)), shading interp
% colormap('hot'), xlabel('Time [sec]'), ylabel('Frequency [Hz]'),
 title('Log of Recorder Spectrogram')
```



Published with MATLAB® R2015b