Homework 2: Gábor Transforms

AMATH 482: Computational Methods for Data Analysis

Rosemichelle Marzan

February 7, 2020

Abstract **– This report demonstrates how Gábor transforms are used to analyze different audio signals. In Part I, we explore the effects of modulating the Gábor transform for filtering audio signal. In Part II, we reproduce the music scores for two instruments based on the spectrograms produced by Gábor filtering.**

1 Introduction and Overview

The aim of this report is to explore how Gábor transforms can be used to analyze different audio signals. Gábor filtering is useful for analyzing how the frequency content of a signal changes with time. In Part I of this report, several spectrograms were produced to analyze the time-frequency domains of a portion of Handel's *Messiah*. These spectrograms demonstrate the effect of filter window widths, oversampling and undersampling, and the type of Gábor filter window used. In Part II, the music scores for two recordings of *Mary had a Little Lamb* played on the piano and a recorder were reproduced using Gábor filtering. Though the same song is played, the different overtone patterns observed in the spectrograms for each instrument accounts for the instrument's unique sound.

2 Theoretical Background

Gábor filtering is a method derived from the fast Fourier transform (FFT) that is essentially a filtering window "sliding" in time. This application of the FFT allows the frequency content of a signal to be analyzed at any time point. We can perform a time-frequency analysis using a spectrogram, which is a color graph that plots frequency on the y-axis and time on the x-axis. Brighter areas on the graph indicate that a greater percentage of a certain frequency is present at a certain time. This filtering technique is used extensively for analyzing speech patterns.

$$\tilde{f}(\tau,\omega) = \int_{-\infty}^{\infty} f(t)g(t-\tau)e^{-i\omega t}dt \tag{1}$$

Eq. 1 is the Gábor Transform, or Short-Time Fourier Transform (STFT), in which τ is the center of the sliding filter window and $g(t-\tau)$ is the filter function. Part I of this report explores different $g(t-\tau)$ functions.

Due to the Heisenberg uncertainty principle, there is a trade-off in time resolution versus frequency resolution. This effect can easily be seen on a spectrogram by modulating filter parameters. A very small window size produces finer horizontal lines (ie. more frequency information) but tend to "bleed" outside of the actual time period where those frequencies are played. On the other hand, a very large window size contains more frequencies (both high and low), which produces finer vertical lines (ie. more time information) that make it difficult to distinguish the frequencies being played for a given time period.

3 PART I. Exploring Spectrograms

3.1 Algorithm Implementation and Development

A 9 second clip of Handel's *Messiah* was loaded from MATLAB's built-in audio file. The time domain was set from 0 to the length of the recording, with n data points based off of the total data points for the signal. For a sampling frequency of 8.192 kHz, 73,113 data points were obtained. Since this is an odd number, the `k` is defined to also contain an odd number of points. The frequency domain, k, is in Hertz units, and so is scaled by 1/L as opposed to 2π/L (note that $\omega = 2\pi f$).

The basic algorithm for generating the spectrograms is to use a `for` loop that slides a filter window along the time domain. A `tslide` vector is created, containing time points from 0 to the length of the recording, at intervals dτ. dτ is how much the filter is translated for each iteration in the loop. The loop contains the filter function (`g`), the filtered signal (`Sg`), and the transform of the filtered signal (`Sgt_spec`). The filter functions, $g(t-\tau)$, used in this assignment were the Gaussian (Eq. 2), the Mexican Hat wavelet (Eq. 3), and the Shannon (step-function) wavelet (Eq. 4). Each were modified to depend on the `a` parameter, which controls the width of the window, and dτ.

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

$$g(t-\tau)_{mexh} = (1-a(t-\tau)^2)e^{-a(t-\tau)^2/2} \tag{3}$$

$$g(t-\tau)_{shann} = u((t-\tau)+a) - u((t-\tau)-a) \tag{4}$$

To filter the signal, the function is multiplied by the time-domain signal. It is important that the FFT step occurs after filtering and not before because the function filters the signal with respect to the time domain. Recall that the power of the Gábor transform is to allow us to focus on one time segment at a time, to analyze how the frequency content changes. The purpose of the FFT step is to generate the frequency-domain signal used to plot the spectrogram.

The spectrogram is plotted using `tslide` to produce the time axis and `ks` for the frequency axis. `tslide` is used instead of vector `t` to avoid matrix disagreement because the filtered signal, `Sgt_spec`, is made up of the same number of rows as `tslide`. When plotting with the shifted `ks` values, `Sgt_spec` must also be `fftshift`-ed (if using non-shifted `k`, `Sgt_spec` must also not be shifted) so that the values line up. `Sgt_spec` is then transposed so that the number of rows matches the length of `tslide` and that the number of columns matches the length of `ks`. The spectrograms were made using the functions `pcolor`, `shading interp`, and `colormap` (see **Appendix A**). Although MATLAB has a built-in spectrogram function, part of the learning objectives for this assignment is to generate our own spectrograms.

3.2 Computational Results

3.2.1 Varying window width

The Gaussian function shows an inverse relationship between a and the function width; thus, the larger the a, the smaller the window width. Decreasing the window width allows higher frequencies to be captured, but loses information for lower frequencies whose wavelength is much larger than the window width.

As predicted by the Heisenberg uncertainty principle, when the window width is too large, we lose temporal resolution. This can be visualized by overlaying onto the signal (in amplitude v. time) a Gaussian function whose width spans the entirety of the audio signal. Essentially, very little of the signal is filtered. Although all of the frequencies "fit" within the filter window, we have little to no information regarding when each of those frequencies occur in time. This phenomena can be observed on a spectrogram by what appear to be simply horizontal lines, which indicate that frequencies bleed into neighboring time windows (Fig. 1, a = 100). Conversely, the more we know about time, the less we know about frequency. This phenomena presents itself as more vertical lines on the spectrogram when the window width is extremely small, which show more frequency overlap (Fig. 1, a = 100,000).

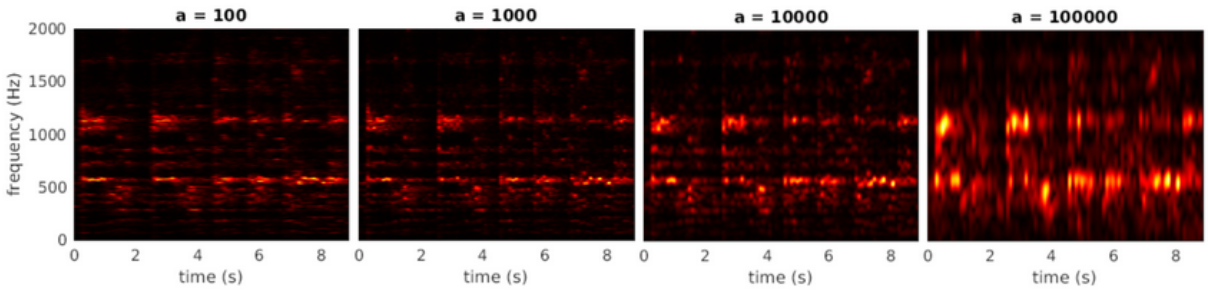

Figure 1: The effect of window size, controlled by parameter a, on frequency resolution with respect to time.

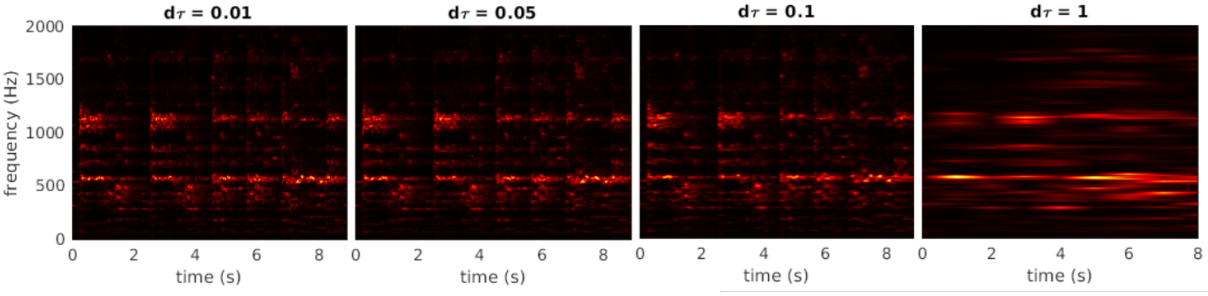

Figure 2: The effect of the translation parameter, τ, on frequency resolution with respect to time.

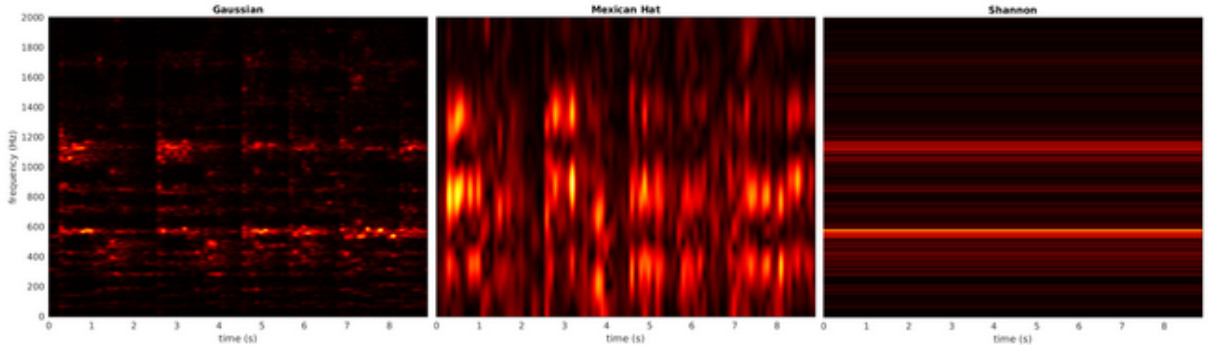

Figure 3: The effect of different filter shapes on frequency resolution with respect to time. From left to right: Gaussian with ideal parameters, $a = 1000$ and $\mathrm{d}\tau = 0.1$, Mexican hat wavelet, and Shannon wavelet also set to the same parameter values.

3.2.2 Varying window translation size

The amount by which the window is translated in time is controlled by τ. Unlike a, τ is directly proportional to the translation size. When the translation size is too large, such as when $\mathrm{d}\tau = 1$ (Fig. 2, we come across the idea of undersampling, which is related to time resolution loss. When $\mathrm{d}\tau = 1$, the spectrogram shows more horizontal streaks.

A quick glance of the *Messiah* spectrograms show that most of the frequencies we hear are around 600 Hz and 1000-1300 Hz range, and are, for the most part, present throughout the entire song. Thus, in the case of this sample of *Messiah*, time resolution may be more important than frequency.

Much smaller $\mathrm{d}\tau$'s were also tested to explore the idea of oversampling. In most cases, oversampling is less detrimental than undersampling when analyzing signals because we can more exactly pinpoint when specific frequencies occur in time. Modulating $\mathrm{d}\tau$ allows us to easily examine a signal at any magnitude of a second.

But for songs, changes in frequency typically do not occur at magnitudes less than 0.05 seconds. At this point, oversampling tends to pose more of a technical/computational concern than an analytical one because computing the data points for the filtered signal at $d\tau = 0.01$, for instance, is computationally expensive (and can take a couple of minutes longer to run than when $d\tau = 0.1$, while also making the program prone to crashing), only to produce a slightly more sharper spectrogram.

3.2.3 Varying filter function

The Gaussian function is appropriate for filtering music signals because it can filter out extremely low and extremely high signals that can be imperceptible to the ear. Fig. 3 is a prime example of how ideal parameters chosen for type of filter function can produce drastically unideal spectrograms for other functions. For all three functions, a was set to 1000 and $d\tau$ to 0.1 which were found to be the ideal parameter values for the Gaussian. The Mexican Hat wavelet filter (Fig. 3, center) shows poor frequency resolution, while the Shannon (step-function) filter (Fig. 3, right) shows poor time resolution.

The Mexican Hat function is similar to the Gaussian in that both have a large, bell-shaped central peak. However, when set to the parameters ideal for the Gaussian, the Mexican Hat function appears to be a very thin peak, which, as demonstrated earlier, results in poorer frequency resolution.

The Shannon function is essentially a square unit pulse that filters in all frequencies within the pulse, without changing the amplitudes of those frequencies (unlike the bell-shaped peaks in the Gaussian and the Mexican Hat). Unlike the previous two functions, a smaller a produces a smaller window width, which in turn produces better time resolution.

4 PART II. Reproducing Music Scores

4.1 Algorithm Implementation and Development

The spectrograms in Part II use the Gaussian window function and are produced using the same algorithm outlined in Part I. The piano and recorder audio files for *Mary had a Little Lamb* were loaded via `audioread`. Since both audio files contain significantly more data points (701,440 for the piano and 627,712 for the recorder) than the *Messiah* file, producing spectrograms for all these points is computationally expensive. An arbitrary cutoff point was defined and is based on the frequency of the highest key on a piano, C8 = 4186 Hz. This point was verified visually to be well above the frequencies of interest (eg. fundamental and 1-3 overtones). The `cutoff` variable was set to be the index of this frequency in the `ks` vectors for both the piano and recorder. Before plotting the spectrogram, `ks` and `Sgt_spec` were both truncated to contain just the positive frequencies up to the cutoff point.

The fundamental frequencies of the notes being played were identified by clicking the 'Data Tips' tool in the figure window then selecting the brightest spots in the spectrogram that coincided with that note. This is because the brighter the color, the more that frequency makes up that time segment. This idea was also used to distinguish between the fundamental frequency and the overtones produced when a note is played.

An online frequency-to-note converter tool was used to reproduce the notes from the frequencies found (`https://newt.phys.unsw.edu.au/music/note/`). The music scores in Fig. 4 were made with Noteflight, an online music notation software.

4.2 Computational Results

4.2.1 Reproducing the music score

Based on the piano and recorder spectrograms, the music score for *Mary had a Little Lamb* was found to be produced by just three notes: C-D-E on the piano and G-A-B on the recorder. Both spectrograms show a similar pattern of notes being played. Although the notes played on the recorder are higher, the frequencies are actually a multiple of the fundamental frequencies played on the piano. For example, the first note played on the piano has a frequency of approximately 336 Hz (E4), and the first note on the recorder has a frequency

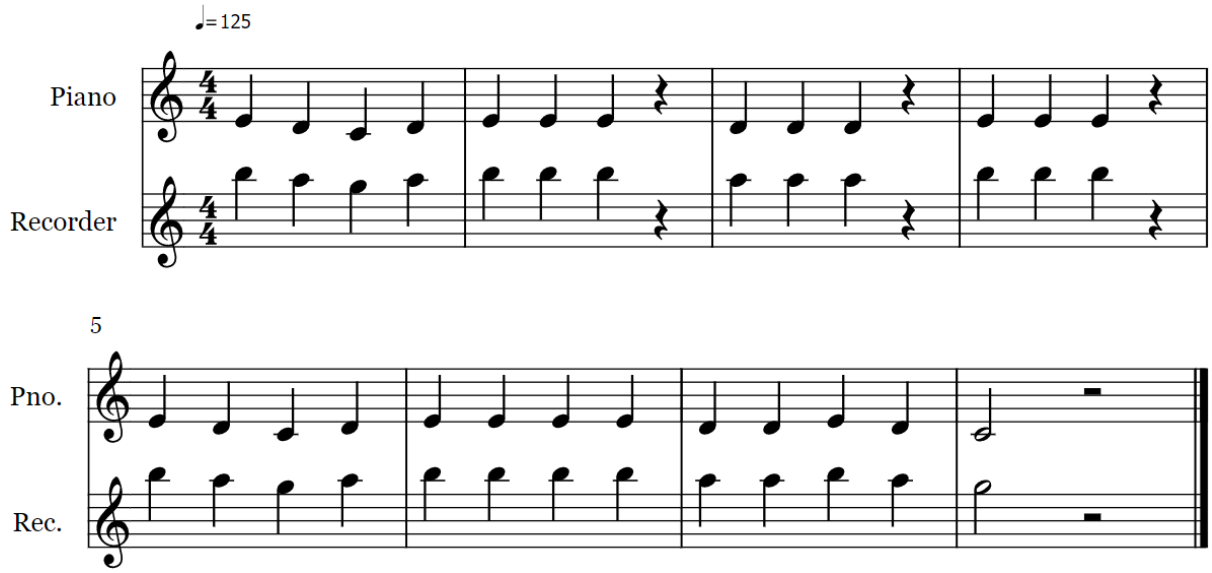

Figure 4: The music scores reproduced for both the piano and the recorder versions of *Mary had a Little Lamb*. Music score generated with Noteflight software.

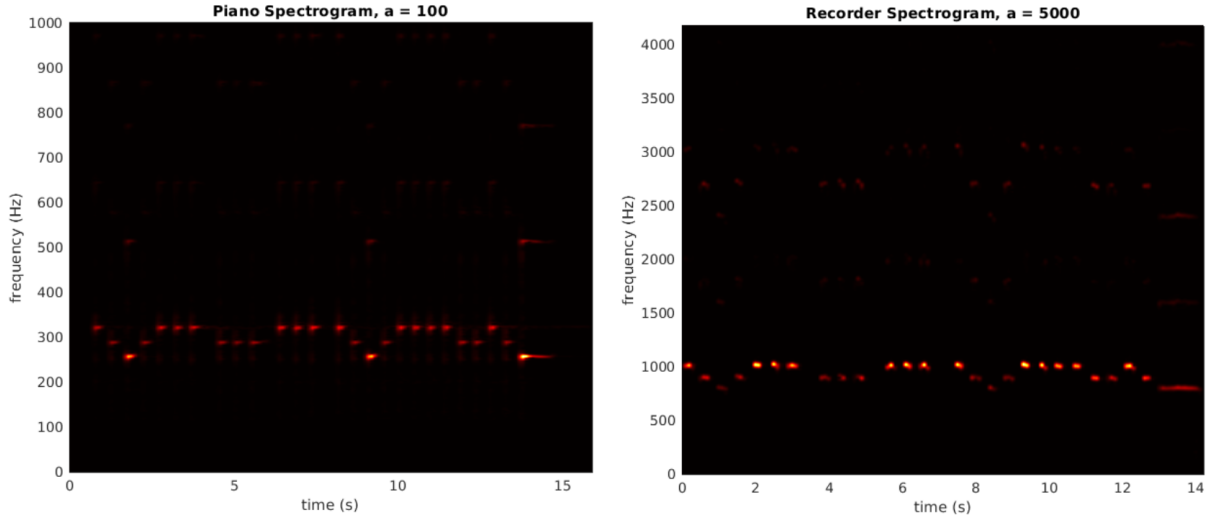

Figure 5: The piano and recorder spectrograms for *Mary had a Little Lamb* filtered to attenuate as much of the overtones as possible. The notes in Fig. 4 are based off of the brighter red/orange/yellow dots, considered to be the fundamental frequencies (ie. notes actually being played). The overtones are the fainter dots present at higher frequencies.

of about 1013 Hz (B5), which is approximately 3x the first piano note frequency. All the recorder notes are 3x higher in frequency than the piano notes.

4.2.2 Difference in overtones

The spectrograms in Fig. 5 were specifically tuned to filter out as much of the overtones as possible, to produce a clean music score. However, faint shadows of the attenuated overtones can still be seen if zoomed in enough. A general comparison between the overtones found in the piano versus the recorder spectrograms

show that the overtones in the recorder recording are more prominent, and a pattern that follows that of the fundamental frequencies is more easily distinguishable. In contrast, the overtones produced in the piano spectrogram could be considered "sparse", with the first overtone, aka $2\omega_0$ ($\sim$523.55 Hz, or C5) produced being the most prominent, whenever C4 is played. When the last note is played, a very faint dot can be seen at the second overtone, aka $3\omega_0$ ($\sim$783.99 Hz, or G5), which as mentioned earlier, is the G note played on the recorder.

When an instrument plays a note, it will produce overtones at each integer multiple of the fundamental frequency (eg. at $2\omega_0$, $3\omega_0$, $4\omega_0$, etc). This is related to the *timbre* of the instrument, which gives the instrument its characteristic sound. However, the intensity of these overtones are not always consistent. For example, as mentioned earlier, a single note from the second overtone in the piano spectrogram is only slightly visible in Fig. 5. The second overtones in the recorder spectrogram are slightly greater in intensity than the first overtones which lie in the 1500 - 2000 Hz range. These unique frequency signatures are what make the piano and recorder sound different, in addition to playing different notes, while playing the same song.

5 Summary and Conclusions

In this report, we investigated how the Gábor transform is used to analyze a signal's frequency content over time. We examined the effect of changing the filter width, the translation size, and the type of filter on the resolutions of both time and frequency of an audio signal. The filter width, controlled by parameter a, produces a better time resolution, but sacrifices frequency resolution when it is wider, while a smaller window produces better frequency resolution at the expense of the time resolution. In windows with a larger translation size, undersampling poses a problem to the time resolution. A smaller translation size produces a crisper spectrogram, which allows us to examine exactly which frequencies are played at what time, but this oversampling of the signal can be computationally expensive. Finally, we found that the type of filter (the shape as defined by the $g(t-\tau)$ function) can be affected by the a and $\mathrm{d}\tau$, parameters differently. When the ideal parameter values for the Gaussian function were applied to other functions, we observed poorer frequency resolution when using the Mexican Hat wavelet function and poorer time resolution when using the Shannon wavelet function.

Gábor filtering the piano and recorder audio files for *Mary had a Little Lamb* allowed us to examine how an instrument's characteristic sound is produced. In addition to the fundamental frequencies (shown as the brighter dots in the spectrogram), faint shadows present at integer multiples of the fundamental frequencies, known as overtones, are also present. The intensity of these overtones is related to the sound the instrument produces. To reproduce the music score for both instruments, the Gábor transform was applied to both recordings then tuned to filter out the overtones, thereby enhancing the frequency resolution of the notes actually played.

6 Appendix A. MATLAB functions used and brief implementation explanation

- **load handel**: Loads a sample of Handel's *Messiah* from MATLAB.
- **pcolor**: Generates a color plot of an m x n matrix based on the given X (m-length) and Y (n-length) vectors as axes.
- **shading interp**: Used so that the spectrogram created with `pcolor` is not just a black window.
- **colormap**: Sets the color palette for `pcolor` plot.
- **num2str**: Converts a numerical value to a string; useful for titles of plots with varying parameters.
- **heaviside**: Generates a heaviside step function.
- **audioread**: Imports audio file and generates a vector of sampled data and the sampling frequency.

7 Appendix B. MATLAB code

```
clear all; close all; clc;

%%% Name: ROSEMICHELLE MARZAN
%%% Course: AMATH 482
%%% Homework 2, Due 2/7/2019

%% PART I: Spectrograms for Handel's 'Messiah'

load handel

% Defining the domain
S = y';                              % signal
L = length(S)/Fs;                    % length of sample
n = length(S);                       % # of data points
t = linspace(0,L,n);                 % vector of time points
k = (1/L)*[0:(n-1)/2 -(n-1)/2:-1]; % vector of frequencies
ks = fftshift(k);

% VARYING WINDOW SIZE (Gaussian)
a = 1000; dtau = 0.1;
tslide = 0:dtau:L;
Sgt_spec = zeros(length(tslide),n);
for j = 1:length(tslide)
    g = exp(-a*(t - tslide(j)).^2);
    Sg = g.*S;
    Sgt = fft(Sg);
    Sgt_spec(j,:) = fftshift(abs(Sgt));
end

pcolor(tslide,ks,Sgt_spec.'),
shading interp
colormap(hot)
ylim([0 2000])
title(['Messiah spectrogram, a = ', num2str(a), ', d\tau = ', num2str(dtau)])
xlabel('time (s)')
ylabel('frequency (Hz)')


% OVERSAMPLING AND UNDERSAMPLING (Gaussian)
a = 1000; dtau = 1;
tslide = 0:dtau:L;
Sgt_spec = zeros(length(tslide),n);
for j = 1:length(tslide)
    g = exp(-a*(t - tslide(j)).^2);
    Sg = g.*S;
    Sgt = fft(Sg);
    Sgt_spec(j,:) = fftshift(abs(Sgt));
end

pcolor(tslide,ks,Sgt_spec.'),
shading interp
colormap(hot)
ylim([0 2000])
title(['Messiah spectrogram, a = ', num2str(a), ', d\tau = ', num2str(dtau)])
xlabel('time (s)')
ylabel('frequency (Hz)')


% MEXICAN HAT WINDOW
a = 10; dtau = 1;
tslide = 0:dtau:L;
Sgt_spec = zeros(length(tslide),n);
for j = 1:length(tslide)
```

```
    % Mexican Hat filter function
    g = (1 - (a*(t - tslide(j))).^2).*exp((-(a*(t-tslide(j))).^2)/2);
    Sg = g.*S;
    Sgt = fft(Sg);
    Sgt_spec(j,:) = fftshift(abs(Sgt));
end

pcolor(tslide,ks,Sgt_spec.'),
shading interp
colormap(hot)
ylim([0 2000])
title(['Messiah spectrogram, Mexican Hat: a = ', num2str(a), ', d\tau = ', num2str(dtau)])
xlabel('time (s)')
ylabel('frequency (Hz)')


% SHANNON WINDOW
a = 0.1; dtau = 0.1;
tslide = 0:dtau:L;
Sgt_spec = zeros(length(tslide),n);
for j = 1:length(tslide)
    % Shannon filter function
    g = heaviside((t - tslide(j)) + a) - heaviside((t - tslide(j)) - a);
    Sg = g.*S;
    Sgt = fft(Sg);
    Sgt_spec(j,:) = fftshift(abs(Sgt));
end

pcolor(tslide,ks,Sgt_spec.'),
shading interp
colormap(hot)
ylim([0 4000])
title(['Messiah spectrogram, Shannon Hat: a = ', num2str(a), ', d\tau = ', num2str(dtau)])
xlabel('time (s)')
ylabel('frequency (Hz)')


%% PART II: Music Scores for 'Mary had a Little Lamb'

% PIANO RECORDING
% defining the domain
[S,Fs] = audioread('music1.wav');
S = S';
L = length(S)/Fs;
n = length(S);
t = linspace(0,L,n);
k = (1/L)*[0:(n/2-1) -n/2:-1];
ks = fftshift(k);

% Gabor filtering
a = 100; dtau = 0.1;
tslide = 0:dtau:L;
Sgt_spec = zeros(length(tslide),n);
for j = 1:length(tslide)
    g = exp(-a*(t - tslide(j)).^2);
    Sg = g.*S;
    Sgt = fft(Sg);
    Sgt_spec(j,:) = fftshift(abs(Sgt));
end

% plotting the piano spectrogram
cutoff = 417301; % where ks frequency = 4186 Hz (arbitrary limit defined by highest piano
    key)
ks = ks(end/2 + 1:cutoff);
Sgt_spec = Sgt_spec(:, end/2 + 1:cutoff);
pcolor(tslide,ks,Sgt_spec.'),
```

```
shading interp
ylim([0 1000])
colormap(hot)
title(['Piano Spectrogram, a = ', num2str(a)])
xlabel('time (s)')
ylabel('frequency (Hz)')


% RECORDER RECORDING
% defining the domain
[S,Fs] = audioread('music2.wav');          % import audio file
S = S';
L = length(S)/Fs;
n = length(S);
t = linspace(0,L,n);
k = (1/L)*[0:(n/2-1) -n/2:-1];
ks = fftshift(k);

% Gabor filtering
a = 5000; dtau = 0.1;
tslide = 0:dtau:L;
Sgt_spec = zeros(length(tslide),n);
for j = 1:length(tslide)
    g = exp(-a*(t - tslide(j)).^2);
    Sg = g.*S;
    Sgt = fft(Sg);
    Sgt_spec(j,:) = fftshift(abs(Sgt));
end

% plotting the recorder spectrogram
cutoff = 373438; % where ks frequency = 4186 Hz (arbitrary limit defined by highest piano
    key)
ks = ks(end/2 + 1:cutoff);
Sgt_spec = Sgt_spec(:, end/2 + 1:cutoff);
pcolor(tslide,ks,Sgt_spec.'),
shading interp
ylim([0 4186])
colormap(hot)
title(['Recorder Spectrogram, a = ', num2str(a)])
xlabel('time (s)')
ylabel('frequency (Hz)')
```