AMATH 582 Homework 2

Kiana Mittelstaedt

February 13, 2020

Abstract

In this report, we will use time-frequency analysis to analyze audio signals. We will construct Gábor filtering windows to slide across three distinct signals. The filtering functions considered are the Gaussian function, the Mexican Hat wavelet, and the Shannon window. By sliding the windowed filters over our data, we capture the time-frequency content through the use of the Fourier transform. We then construct spectrograms of our signals and analyze the effect of varying the window width.

1 Introduction and Overview

In time-frequency analysis, we can capture both the time and frequency components of a given signal. In this paper, we are going to analyze three different audio signals; namely, a 9-second clip of Handel's Messiah, a 16-second clip of *Mary had a little lamb* played on the piano, and a 14-second clip of *Mary had a little lamb* played on the recorder. Audio signals are *non-stationary*, i.e., the frequency content evolves in time. To capture this evolution and effectively extract the signal information, we will use windowed transforms.

To study our three music clips, we will implement the Gábor windowed transform. The Gábor transform decomposes the audio signal into small sections of time, applies a filter to the isolated time frame, and calculates the spectral content using the Fourier transform in that time window. The frequency components are mapped out by shifting the window across the entire time domain. As a result, we can localize in both time and frequency. We produce spectrograms of our audio signals using the data we collected in the windowed Fourier transforms and compare the result of using different filtering functions and window widths. The spectrograms allow us to visualize our song clips in both the time and frequency domains.

2 Theoretical Background

2.1 Gábor Transform

The Fourier transform takes a signal in time and captures the signal's frequency components, but it does not render any information about when these frequencies occurred in time. As a result, the Fourier transform is best for *stationary* signals, i.e., signals whose spectral content is fixed in time. However, most signals are *non-stationary*, meaning that the signal changes in time and evolves in frequency.

To analyze *non-stationary* signals, we will use the Gábor transform. This transform modifies the kernel of the Fourier transform as follows:

$$g_{t,\omega} = e^{i\omega\tau} g(\tau - t). \tag{1}$$

It follows that the *Gábor transform* is defined as:

$$G[f](t,\omega) = \tilde{f}_g(t,\omega) = \int_{-\infty}^{\infty} f(\tau)\bar{g}(\tau - t)e^{-i\omega\tau}d\tau, \tag{2}$$

where the bar represents the complex conjugate of the function g. The function $g(\tau - t)$ acts as a time filter for localizing the signal over a time window, i.e., it provides localization in time and frequency. By integrating over τ, the time-filtering window will slide over the entire signal.

When the time and frequency domains are discretized to the sample points

Time series vs. Fourier Transform vs. Gábor Window vs. Wavelets

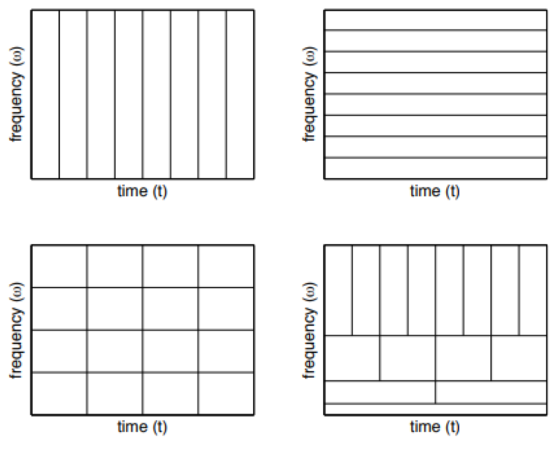


Figure 1: Graphical comparison between time series analysis, Fourier analysis, Gábor analysis and wavelets.

$$\nu = m\omega_0$$
$$\tau = nt_0$$

we use the discrete definition of the Gábor transform:

$$\tilde{f}(m,n) = \int_{-\infty}^{\infty} f(t)\bar{g}_{m,n}(t)dt, \tag{3}$$

where m and n are integers and ω_0, $t_0 > 0$ are constants.

To obtain great localization in both time and frequency, the signal should be over-sampled, i.e., $0 < t_0, \omega_0 < 1$. The Gábor discretization will be unable to reproduce the signal if $\omega_0, t_0 > 1$ because the signal will be under-sampled.

In this report, we consider a Gaussian function for the Gábor window. The Gaussian function is given by

$$g(t) = e^{-a(t-b)^2}, \tag{4}$$

centered at b with width a.

2.2 Wavelets

Wavelets are a simple modification of the Gàbor method: the filter window varies in size. Doing so allows for the low-frequency components to be extracted through a large scaling window, and the high-frequency components through a narrower window (from poor time resolution to great time resolution). This method is thus capable of collecting excellent time *and* frequency resolution.

The wavelet we consider in this report is the Mexican hat wavelet. Defined below is the Mexican hat wavelet, $\psi(t)$, with its transform, $\hat{\psi}(\omega)$:

$$\psi(t) = (1-t^2)e^{\frac{-t^2}{2}} = -\frac{d^2}{dt^2}e^{\frac{-t^2}{2}} = \psi_{1,0}$$
$$\hat{\psi}(\omega) = \hat{\psi_{1,0}}(\omega) = \sqrt{2\pi}\omega^2 e^{\frac{-\omega^2}{2}}.$$

This function is a second moment of a Gaussian in the frequency domain and thus the Mexican hat wavelet has great time and frequency localization. The filter for the Mexican hat wavelet we utilize in this report is:

$$g(t) = (1-bt)e^{-a(t-c)^2}, \tag{5}$$

centered at c with width a and maximum negative value b.

Finally, we consider the Shannon filter window, which is a step function with value one inside the transmitted band and zero else. For varying filter window widths, the difference between the Gaussian window and the Shannon window is small.

Figure 1, taken from Nathan's book, summarizes the difference in resolution between time series analysis, Fourier analysis, Gábor analysis, and wavelets. Using time series, the signal is resolved well in time, but not at all in frequency. Fourier analysis is the opposite: there is good frequency resolution, but no time resolution. The Gábor transform provides time and frequency resolution at the expense of some level of accuracy in each domain. However, the Gàbor transform limits the level of time-frequency resolution because there is a fixed window size; the wavelet transforms allow the scaling window to change.

2.3 Spectrograms

To visualize our three audio signals in both the time and frequency domains, we will use a spectrogram. When we apply the Gábor transform, we collect different frequency content for each time window. This frequency data is collected into a matrix that can be plotted as a spectrogram that shows the signal frequency versus time. In this report, we will produce several spectrograms of Handel's Messiah and *Mary had a little lamb*. We compare the resolution as we vary the window width and apply different Gábor function filters.

3 Algorithm Implementation and Development

In this paper, we will produce spectrograms for our three audio signals: a 9-second clip of Handel's Messiah, a 16-second clip of *Mary had a little lamb* played on the piano, and a 14-second clip of *Mary had a little lamb* played on the recorder. We will vary the width and the distribution type of the Gábor window and analyze the result on the spectrogram produced. The numbered list below describes the over-arching methods we utilized in this project.

1. Load the audio files into MATLAB. Extract the signal (given as a column vector) and transpose it and store as a row vector.
2. Discretize the time and frequency domains. Because we are analyzing audio data, we want to interpret k as frequency in hertz (Hz), not wave number. As such, we do not need to scale our frequency mode by 2π. Fourier transform shift our frequency mode.
3. Initialize a time slide that governs how the filter window moves across the signal.
4. Define the window function. We consider the Gaussian window, Mexican hat wavelet, and Shannon window.
5. Inside a loop, multiply our audio signal by the specified filter window. Then, Fourier transform this filtered signal.
6. Store the absolute value of the shifted Fourier transform of our signal (as a vector) into the spectrogram matrix. Note that we need to shift our transformed signal back into our domain of interest because the Fourier transform assumes we are working on a 2π-periodic domain.
7. When the loop terminates, plot the spectrogram matrix with time on the x-axis and frequency (in Hz) on the y-axis.

Using the process outlined above, we will vary the window width and functions to analyze how the spectrogram resolution is affected.

4 Computational Results

4.1 Part I: Handel's Messiah

First, we use time-frequency analysis to analyze a portion of Handel's Messiah. The signal from the portion of music analyzed and its Fourier transform are shown in Figure 2. This song clip is 8.9249 seconds long and

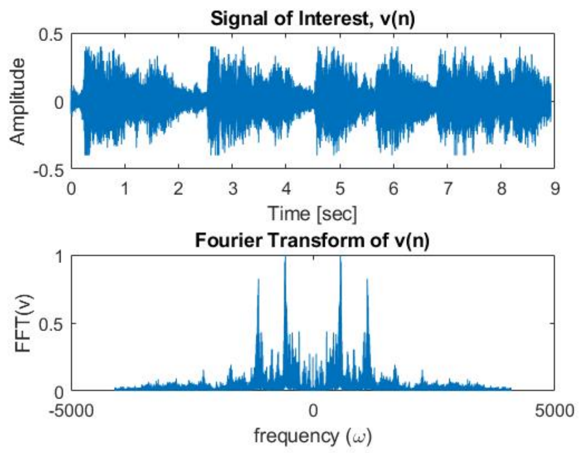


Figure 2: Signal from the 9-second clip of Handel's Messiah

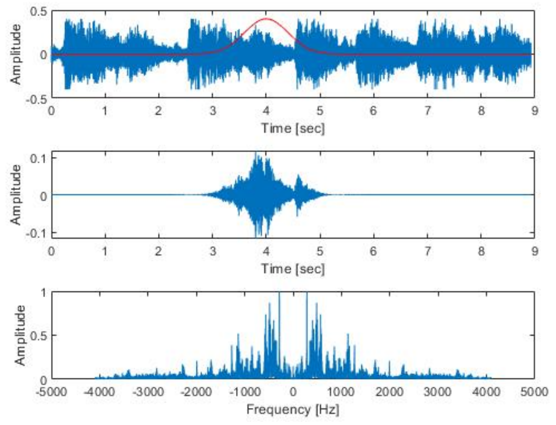


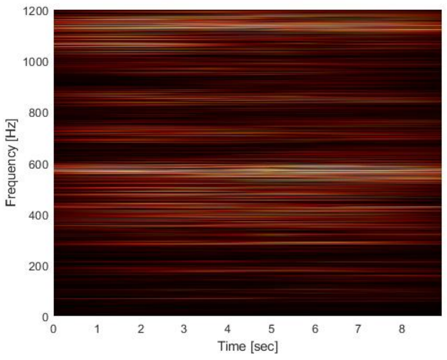
Figure 3: Top: Signal with sliding Gábor window. Middle: Signal multiplied by the Gaussian function. Bottom: Fourier transform.

is discretized into $73,113$ samples, producing a sampling period of $1.2e^{-4}$ seconds. Our sampling frequency is 8192 Hz (cycles per second).

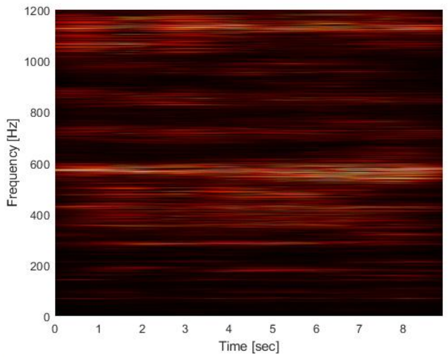
We construct a Gaussian Gábor filter window and slide it across our signal. At each time slide, we multiply our signal by the Gaussian filter. Then, we Fourier transform that window of data to collect the frequency content. Figure 3 shows this process. The frequency content collected from the sliding window (at each time) is stored as a vector and added to a matrix. Once the slide is complete, this matrix is plotted as a spectrogram.

Figure 4 shows spectrograms for Handel's Messiah using different window widths a with fixed time slide Δt= 0.1 seconds. Note that the frequency range for the spectrograms of Handel's Messiah is from $0-1200$ Hz, the range of human vocals. We see that with a small window width (for example, $a = 500$), we lose location in frequency, but gather great time localization. Conversely, a large widow width (for example, $a = 0.2$), we produce great frequency resolution at the expense of localization in time. This accurately describes the trade-off between time and frequency resolution as a result of applying the Gàbor method; regardless of window width, we will always be throwing away information.

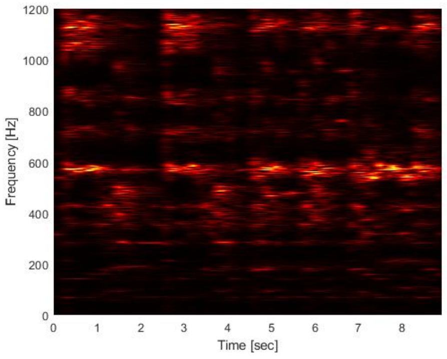
We also consider varying the size of the time-slide, i.e., whether or not we over-sample or under-sample. To over-sample, we use very small translations of the Gàbor window, and to under-sample, we use very large translations of the Gàbor window. The window width was held constant at $a = 10$, but we tested Δt=0.05s



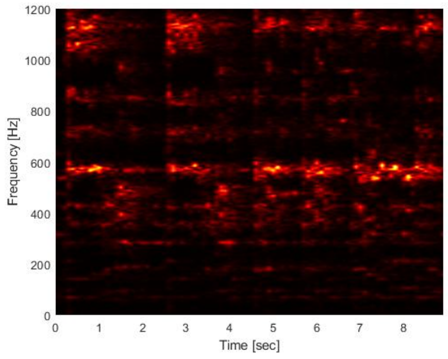
(a) a=0.2, Δt=0.1s



(b) a=1, Δt=0.1s

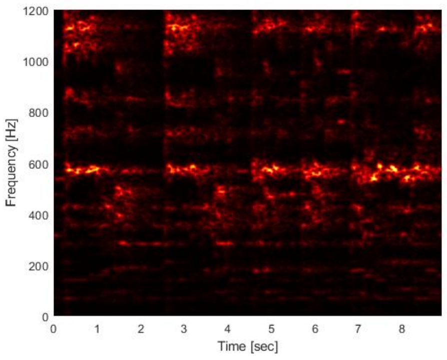


(c) a=50, Δt=0.1s

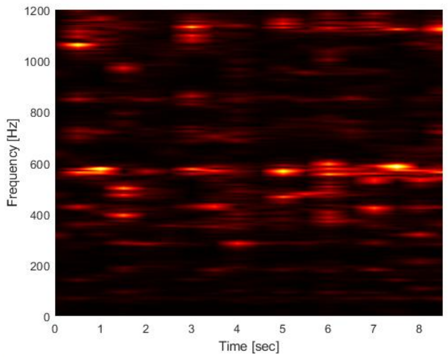


(d) a=500, Δt=0.1s

Figure 4: Spectrograms showing the Gaussian Gàbor window filter of varying widths a for Δt=0.1s.

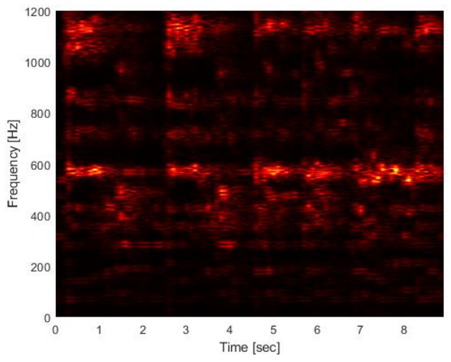


(a) a=500, Δt=0.05s

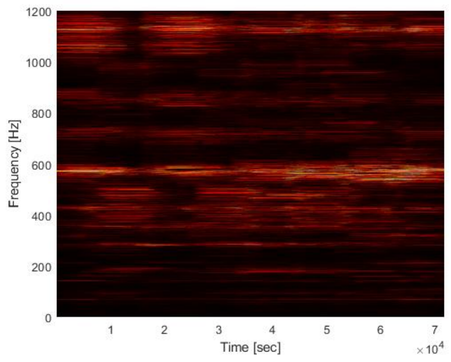


(b) a=500, Δt=0.5s

Figure 5: Spectrograms showing the Gaussian Gàbor window filter of constant width $a = 500$ for varying time slides Δt=0.05s and Δt=0.5s.



(a) Mexican hat wavelet



(b) Shannon window

Figure 6: Spectrograms produced from the Mexican hat wavelet and the Shannon window.

(over-sampled) versus Δt=0.50s (under-sampled). Figure 5 shows these results. We see that when the signal is under-sampled, we lose frequency resolution.

The same process of was applied using the Mexican hat wavelet and the Shannon window and the spectrograms are shown in Figure 6. The spectrogram produced from using the Mexican hat wavelet looks strikingly similar to that of the Gaussian filter window, but it appears to have slightly better resolution in frequency. The Shannon window lost frequency resolution, but varying the width of the step function as well as decreasing the slide size could help recover the lost resolution.

From the spectrogram of Handel's Messiah, especially in Figure 4d, we can slightly deduce the music score of "Hallelujah". There are several frequencies present because the clip has both female and male vocalists, and the sharp cut-offs in time correspond to when the choir is singing loudly.

4.2 Part II: Mary Had a Little Lamb

In this section of the project, we use Gàbor filtering to produce the music score for *Mary had a little lamb*. We have two recordings of this piece, one played on the piano and the other on the recorder, and we will compare the spectrograms produced.

The recording of *Mary had a little lamb* on the piano is about 16 seconds long and we discretized it into $701,440$ samples, producing a sampling period of $2.3e^{-5}$ seconds. Our sampling frequency is $43,840$ Hz (cycles per second). The clip as played on the recorder is about 14 seconds long and we discretized it into $627,712$ samples, producing a sampling period of $2.2e^{-5}$ seconds. Our sampling frequency is $44,837$ Hz.

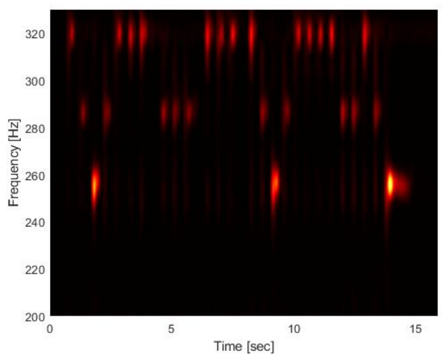
The spectrograms of *Mary had a little lamb* are shown in Figure 7. Note that the frequency range (in Hz) of the piano is different than the recorder. As such, the spectrogram of the piano will range from $200-320$ Hz, whereas the spectrogram for the recorder will range from $700-1,100$ Hz.

As mentioned, the original audio files provided were around 15 seconds, but we edited the clips to only contain the first half of *Mary had a little lamb*. Doing so allowed us to gather better time and frequency resolution; the full recording required a lot of computing power and we could not filter as well. The spectrogram showing the first half of the song is shown in Figure 8.

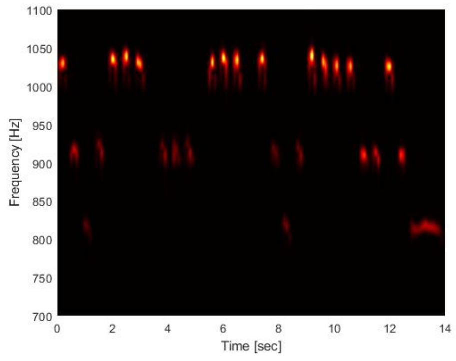
We see from the spectrograms the music score for *Mary had a little lamb*. The song is composed of three notes and we can clearly see these in the spectrogram. We can also observe, through time-frequency analysis, the difference between the music produced by a piano and a recorder.

The 'timbre' of an instrument describes its capability of producing multiple frequencies simultaneously, i.e., the overtones generated by the instrument for a center frequency. More specifically, if the musician plays a note at ω_0, the instrument will generate overtones at $2\omega_0$, $3\omega_0$, and so on.

Figure 8 clearly shows the difference in 'timbre' between the piano and the recorder. The recorder is capable of producing notes at only one center frequency, hence why the spectrogram only shows frequencies of about 725, 825, and 925 Hz. The spectrogram of the piano recording, on the other hand, shows the 'timbre' of the piano. There center frequencies are still the strongest, and occur at about 230, 260, and 290

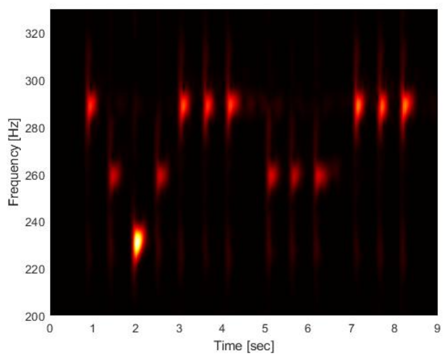


(a) Piano

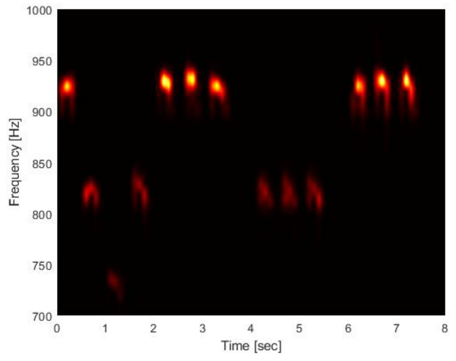


(b) Recorder

Figure 7: Spectrograms comparing frequencies from the piano and the recorder using Gàbor filtering.



(a) Piano



(b) Recorder

Figure 8: Spectrograms with better resolution, as a result of clipping the recordings.

Hz, but there are clearly overtones shown as well; there are frequencies present in the spectrogram that are not the same as the central frequency (shown above and below the central frequency notes as a line).

5 Summary and Conclusions

In this report, we used Gàbor filtering time-frequency analysis techniques to analyze several clips of music. In Part I, we use Handel's Messiah and compare three different filter windows: the Gàbor Gaussian window, the Mexican hat wavelet, and the Shannon window and their resulting spectrograms. Furthermore, we showed the effect of varying the window width or over-sampling and under-sampling on the spectrogram produced. In Part II, we compared the spectrograms of a recording of *Mary had a little lamb*, one played on the piano and the other on the recorder. We produced the music score for the song and discussed the 'timbre' of an instrument; the piano is able to generate overtones while the recorder is not.

The code used to produce the spectrograms in this report can be found on my Github Repository, linked: $https://github.com/kiana-mittelstaedt/AMATH582$

Appendix A MATLAB Functions

Here is a list of the important MATLAB functions I used with a brief implementation explanation of each.

- `[y,Fs]=audioread(FILENAME)` reads an audio file by the string scalar FILENAME and returns the sampled data in y and the sample rate Fs, in Hertz.
- `audioplayer(y,Fs)` creates an audioplayer object for signal y with sample rate Fs.
- `fftn(X)` returns the N-dimensional discrete Fourier transform of the N-dimensional array X.
- `fftshift(X)` is the FFT shift. This function swaps "half-spaces" of X along each dimension for N-dimensional arrays.
- `pcolor(X,Y,C)` makes a pseudocolor or "checkerboard" plot of the matrix C on the grid defined by the vectors X and Y.
- `playblocking(OBJ)` plays, from the beginning, the audio sample in an audioplayer object.
- `round(X)` rounds each element of X to the nearest integer.

Appendix B MATLAB Code

Code for Part I: Handel's Messiah

```
clear all; close all; clc;

%% Signal creation
load handel % load file
v=y'/2; % this is our signal, transpose to get column vector
n=length(v); % number of samples = 73,113
L=n/Fs; % length of signal

t=(1:n)/Fs; % time vector

k=(1/L)*[0:(n-1)/2 -(n-1)/2:-1]; % k for odd number of points
ks=fftshift(k); % shifted frequency domain

tslide=0:0.5:L; % sliding window from 0 to L with time step 0.05
```

```
%% plot signal
vt=fft(v);

figure(1)
subplot(2,1,1)
plot((1:length(v))/Fs,v); % plot of our signal
set(gca,'Fontsize',[12]), xlabel('Time [sec]'), ylabel('Amplitude')
xlim([0 9]),title('Signal of Interest, v(n)');
subplot(2,1,2)
plot(ks,abs(fftshift(vt))/max(abs(vt)));
set(gca,'Fontsize',[12]), xlabel('frequency (\omega)'), ylabel('FFT(v)')
title('Fourier Transform of v(n)')
vt=fft(v);

figure(1)
subplot(3,1,1)
plot((1:length(v))/Fs,v)
set(gca,'Fontsize',[14]), xlabel('Time [sec]'), ylabel('Amplitude')
xlim([0 9]);

subplot(3,1,2)
plot(ks,abs(fftshift(vt))/max(abs(vt)));
set(gca,'Fontsize',[14])
xlabel('frequency (\omega)'), ylabel('FFT(v)')

p8 = audioplayer(v,Fs); % Handel's song
playblocking(p8);

%% plot signal with filter
figure(2)
width=[0.2 1 5]; % show different width (window size)
for j=1:3
    g=0.4*exp(-width(j)*(t-4).^2); % make a gaussian
    subplot(3,1,j)
    plot(t,v), hold on % plot signal in time
    plot(t,g,'r','Linewidth',[1.5]) % plot filter
    set(gca,'Fontsize',[14]) % plot together
    ylabel('Amplitude'); xlabel('Time [sec]')
    xlim([0 9]);
end

%% mexican hat window
a=10;b=10; % set widths
g2=0.4*(1-b*(t-4).^2).*exp(-a*(t-4).^2); % create filter
plot(t,v), hold on
plot(t,g2,'r','Linewidth',[1.5])
set(gca,'Fontsize',[14])
ylabel('Amplitude'); xlabel('Time [sec]')
xlim([0 9]);


%% slide window
a=500; b=1000; % widths
vgt_spec=[]; % spectrogram data matrix
```

```
for j=1:length(tslide)
    g=exp(-a*(t-tslide(j)).^2); % Gaussian filter
    g=(1-b*(t-tslide(j)).^2).*exp(-a*(t-tslide(j)).^2); % mexican hat
    vg=g.*v; % multiply signal by filter
    vgt=fft(vg); % fourier transform
    vgt_spec=[vgt_spec; abs(fftshift(vgt))]; % build spectrogram matrix

    subplot(3,1,1), plot(t,v,t,g,'r')
    xlabel('Time [sec]'); ylabel('Amplitude');
    subplot(3,1,2), plot(t,vg)
    xlabel('Time [sec]'); ylabel('Amplitude');
    subplot(3,1,3), plot(ks,abs(fftshift(vgt))/max(abs(vgt)))
    xlabel('Frequency [Hz]'); ylabel('Amplitude');
    drawnow
    pause(0.1)
end

%% spectogram
% spectrogram - k content stacked together
figure(5)
pcolor(tslide,ks,vgt_spec.'), shading interp
%set(gca,'Ylim',[100 600],'Fontsize',[14])
xlabel('Time [sec]'), ylabel('Frequency [Hz]')
ylim([0 1200]);
colormap(hot)

%% shannon filter
a=5000; % width
len=5; % length of slide
vgt_spec_2=[]; % spectrogram data matrix
overlap=round(a/len); % set center of shannon window
slide=1:overlap:length(t); % where to slide
for j=1:length(slide)
    shan_filt=shan_step(t,slide(j),a); % call step function
    vg=shan_filt.*v; % multiply signal by filter
    vgt=fft(vg); % fourier transform
    vgt_spec_2=[vgt_spec_2; abs(fftshift(vgt))]; % build spectrogram matrix
end

%spectrogram
figure(6)
pcolor(slide,ks,vgt_spec_2.'), shading interp
%set(gca,'Ylim',[100 600],'Fontsize',[14])
xlabel('Time [sec]'), ylabel('Frequency [Hz]')
ylim([0 1200]);
colormap(hot)
```

Code for Part II: *Mary had a little lamb*, piano

```
clear all; close all; clc;

tr_piano=16;
y=audioread('music1.wav');
Fs=length(y)/tr_piano;
```

```
plot((1:length(y))/Fs,y);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Mary had a little lamb (piano)'); drawnow
p8=audioplayer(y,Fs); playblocking(p8);

P=y'/2;
n=length(P);
L=n/Fs; % 16 time units

t=(1:n)/Fs;
k=(1/L)*[0:(n/2-1) -n/2:-1]; % k for even number of points
ks=fftshift(k); % shifted frequency domain

tslide=0:0.15:L; % sliding window from 0 to L with time step 0.15

%% slide window
a=250;
Pgt_spec=[]; % spectrogram data matrix

for j=1:length(tslide)
    g=exp(-a*(t-tslide(j)).^2); % Gaussian filter
    Pg=g.*P; % multiply signal by filter
    Pgt=fft(Pg); % fourier transform
    Pgt_spec=[Pgt_spec; abs(fftshift(Pgt))];

    subplot(3,1,1), plot(t,P,t,g,'r')
    xlabel('Time [sec]'); ylabel('Amplitude');
    subplot(3,1,2), plot(t,Pg)
    xlabel('Time [sec]'); ylabel('Amplitude');
    subplot(3,1,3), plot(ks,abs(fftshift(Pgt))/max(abs(Pgt)))
    xlabel('Frequency [Hz]'); ylabel('Amplitude');
    drawnow
    pause(0.1)
end

%% spectogram
% spectrogram - k content stacked together
figure(5)
pcolor(tslide,ks,Pgt_spec.'), shading interp
%set(gca,'Ylim',[100 600],'Fontsize',[14])
xlabel('Time [sec]'), ylabel('Frequency [Hz]')
ylim([200 330]);
colormap(hot)
```

Code for Part II: *Mary had a little lamb*, recorder

```
clear all; close all; clc;

tr_rec=14;
y=audioread('music2.wav');
Fs=length(y)/tr_rec;
plot((1:length(y))/Fs,y);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Mary had a little lamb (recorder)'); drawnow
p8=audioplayer(y,Fs); playblocking(p8);
```

```
R=y'/2;
n=length(R);
L=n/Fs; % 14 time units

t=(1:n)/Fs;
k=(1/L)*[0:(n/2-1) -n/2:-1]; % k for even number of points
ks=fftshift(k); % shifted frequency domain

tslide=0:0.1:L; % sliding window from 0 to L with time step 0.1
%% slide window
a=500;
Rgt_spec=[]; % spectrogram data matrix

for j=1:length(tslide)
    g=exp(-a*(t-tslide(j)).^2); % Gaussian filter
    Rg=g.*R; % multiply signal by filter
    Rgt=fft(Rg); % fourier transform
    Rgt_spec=[Rgt_spec; abs(fftshift(Rgt))];

    subplot(3,1,1), plot(t,P,t,g,'r')
    xlabel('Time [sec]'); ylabel('Amplitude');
    subplot(3,1,2), plot(t,Pg)
    xlabel('Time [sec]'); ylabel('Amplitude');
    subplot(3,1,3), plot(ks,abs(fftshift(Pgt))/max(abs(Pgt)))
    xlabel('Frequency [Hz]'); ylabel('Amplitude');
    drawnow
    pause(0.1)
end

%% spectogram
% spectrogram - k content stacked together
figure(5)
pcolor(tslide,ks,Rgt_spec.'), shading interp
%set(gca,'Ylim',[100 600],'Fontsize',[14])
xlabel('Time [sec]'), ylabel('Frequency [Hz]')
ylim([700 1100]);
colormap(hot)
```