HW2: Gábor Transforms

AMATH 482 | Celeste Becker

Abstract

This report has two parts. Both are focused on time-frequency analysis using Gabor Transforms to build spectrograms. Part 1, we will analyze a music sample and explore how changing different properties of the Gabor Transform effects the spectrogram. Additionally, we will see how a Mexican hat wavelet filter effects the spectrogram.

In Part 2, we will analyze music samples of two different instruments playing the same song. The goal is to determine what notes are being played, their order, and the relative duration of each note, and filter out overtones for each music sample.

Sec. I. Introduction and Overview

Part 1: We will analyze a small sample of music from Handel's Messiah. To get resolution in both the time and frequency domains, we will build a spectrogram using the Gabor Transform. We will explore how changing the window width of the Gabor Transform effects the spectrogram. In addition, we will explore the idea of Oversampling and Undersampling. Then we will see how using a different Gabor Window effects the spectrogram by using a Mexican hat wavelet instead of a Gaussian window.

Part 2: Using a spectrogram built with the Gabor window, we will analyze two different music samples of a piano and a recorder both playing the same song, Marry Had A Little Lamb. By creating spectrograms with the right ratio of time-frequency resolution, we can tell what notes are played using a frequency to music note reference, and their relative duration. Overtones are integer multiples of each note's frequency. To make sure that we are looking at the correct frequency values, we will use a gaussian filter to filter out the overtones in the music samples. Once the samples are analyzed, we can explore the instrument's similarities and difference.

Sec. II. Theoretical Background

The main mathematical methods used in this program are the *Gabor Transform,* and the *Mexican hat wavelet.* In addition to *Oversampling and undersampling* a signal. The information below is referenced from sections 13.4 through 13.8 in Chapter 13 of *Data-Driven Modeling & Scientific Computation: Methods for Complex Systems & Big Data* by J. Nathan Kutz.

Gábor Transform: The Gabor Transform, or the short-time Fourier Transform (STFT), captures the time-frequency content of a signal. It works by using a kernel defined as:

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

The function $g(\tau - t)$ localizes a signal over a specific window of time, acting as a time filter. Then the *Gabor Transform* (STFT) is defined as:

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

The time filtering window is centered at τ. The width of the filter is equal to a. $\bar{g}$ is the complex conjugate. The integration over τ is what slides the time-filtering window down the entire signal in order to pick out the frequency information at each instant of time. Note that the Gabor method loses some time and frequency resolution in order to give both time and frequency resolution simultaneously. To apply the Gabor method, we must use a discrete time and frequency domain. The discrete version of $g_{t,\omega}(\tau)$ is then:

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

And the discrete Gabor Transform is:

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

The Gabor Transform is limited by the time filtering. The time window filters out the time behavior of the signal that has a window centered at τ with a width equal to a. This means that when looking at the spectral content of this window, any portion of the signal that has a wavelength longer than the width of the Gabor Window will be lost. This leads to the Heisenberg uncertainty principal: if you have more resolution in one domain (either frequency or time), you will have less resolution in the other.

Oversampling: Using very small translations of a wide Gabor window. If $0 < t_0, \omega_0 < 1$, this causes the Gabor frames (windows) to overlap, creating good resolution of the signal in the frequency domain, and loosing time domain information.

Undersampling: Using very large translations with a narrow Gabor window. If $t_0, \omega_0 < 1$, this causes the Gabor frames to not overlap, loosing frequency signal information and making it incapable of reproducing the input signal.

Mexican Hat Wavelet: The Mexican Hat Wavelet is a function that can be used for time-frequency analysis. It still uses the same main idea of translating a short-time window and scaling the window to capture finer time resolution. Its name comes from the shape of the wavelet.

The Mexican Hat Wavelet is defined as:

$$\psi(t) = (1 - t^2)e^{-t^2/2}$$

Using this wavelet in application, we set a scaling factor to a constant value, creating a fixed window size. This imposes a fundamental limitation on the level of time-frequency resolution that can be obtained.

Sec. III. Algorithm Implementation and Development

Each section labeled here corresponds to the commented section with the same name in *Appendix B. Matlab Codes* at the end of this report. Code reference lines are also included. This program has been split up into three files. Each section starting with PART is a different file.

PART 1 Set Up - (lines 1-14): First load in the music sample. We set the signal S equal to the amplitude of the music sample, and define the discrete frequencies K. We multiply K by (1/L) to convert the frequencies to Hertz. L is the length of the time of the sample.

Spectrograms for Varying Gabor Widths - (lines 15-46): Vector a_vec contains different Gabor filter width values. Here, we define a Gabor Filter window, and loop through the different filter widths creating four different spectrogram plots each with a different filter width. The Gabor Filter is defined as: $g(t) = e^{-a(t-tslide(j))}$ with t = time, **a** = width, and tslide(j) is the discrete time vector of the signal.

Exploring Oversampling and Undersampling - (lines 47-92): Set **a** to a large number, making a very wide filter. Then we slide the filter in very small increments in time to Oversample the music sample. For Undersampling, we set **a** to a very small number, making a narrow filter width. Then we slide the filter in very large time increments. Both spectrograms are plotted.

Mexican Hat Wavelet - (lines 93-115): A Mexican hat wavelet is defined: $\psi(t) = a\left(1-(t-tslide(j))^2\right)e^{-(t-tslide(j))^2/2}$ and used as the filter for time-frequency analysis in the spectrogram. The spectrogram is plotted.

PART 2 (Piano) Set Up - (lines 1-14): Load in the music sample, and define key constants used throughout the rest of the program. We set the signal S equal to the amplitude, and with L as the length of time, we scale the frequencies K by (1/L) to convert to Hertz.

Spectrogram - (lines 15-57): Discrete time vector, tslide, slides the Gabor filter through the signal. Sgt_spec stores the spectrogram data. A gaussian filter is defined to filter out overtones. Then, we define the Gabor filter. To compute the spectrogram, we multiply the signal by the Gabor Filter, and then take the Fourier transform. Then we multiply the Fourier transformed signal by the gaussian filter, and then store the spectrogram data. Then we plot the spectrogram.

PART 2 (Recorder) Set Up - (lines 1-20): This follows the exact same logic as PART 2 (Piano) set up. Just using the Recorder music sample.

Spectrogram - (lines 21-60): This also follows the exact same logic for the Spectrogram in PART 2 (Piano) instead just using the Recorder music sample.

Sec. IV. Computational Results

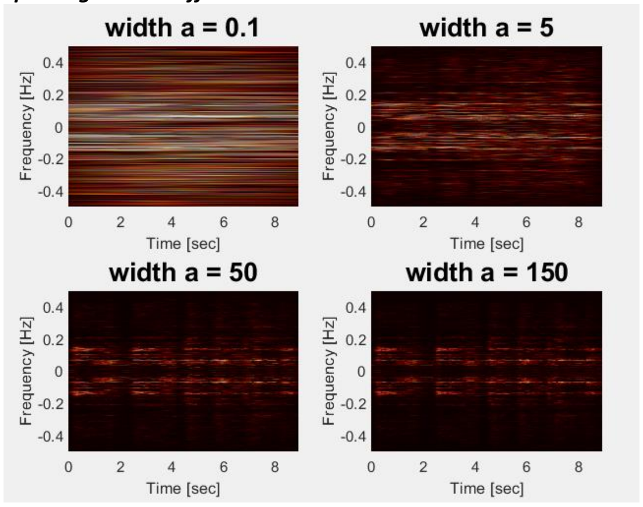
Part 1: ***Spectrogram 1:*** The Handel Music sample analyzed with several different Gabor filter widths. Because the scaling width parameter **a** is located in the exponential, as **a** increases, the width of the Gabor window decreases, and the frequency resolution decreases as the time resolution increases. The best frequency resolution is at width **a** = 1, and the best time resolution is at width **a** = 150.

Spectrogram 2: Built with the Mexican Hat Wavelet filter. There is much better frequency resolution than time resolution, and it looks a little bit Oversampled.

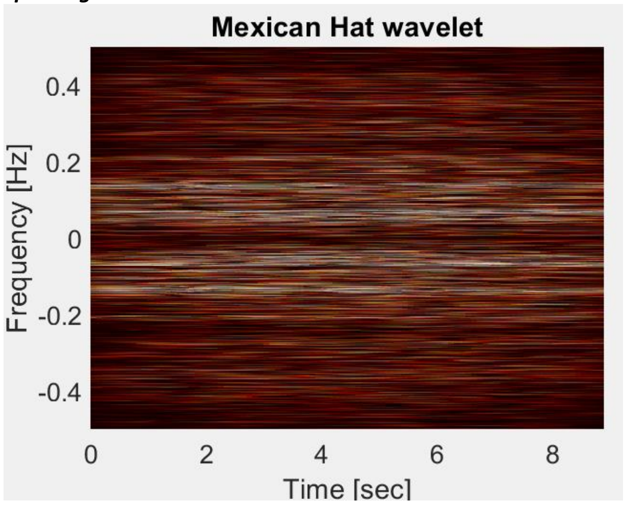
Spectrogram 3: Undersampling by using a very narrow window and sliding using large time increments. We can see that there is a lot of frequency data lost.

Spectrogram 4: Oversampling by using a very wide Gabor window, and sliding using very small time increments. This causes lots of overlap in the Gabor window. This causes a blurring in resolution of frequency, and basically no time resolution.

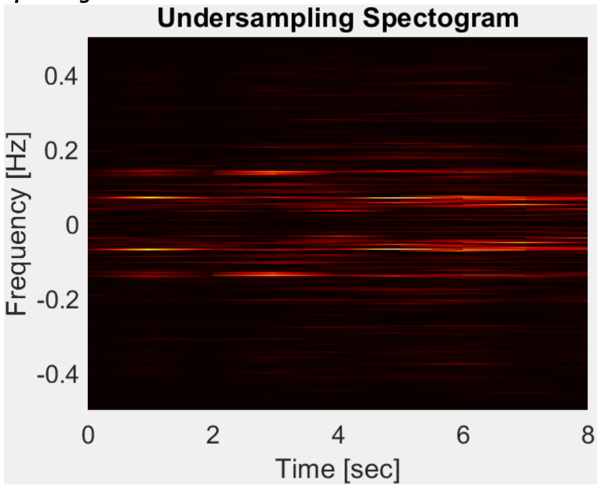
Spectrogram 1 - Different Gabor Widths



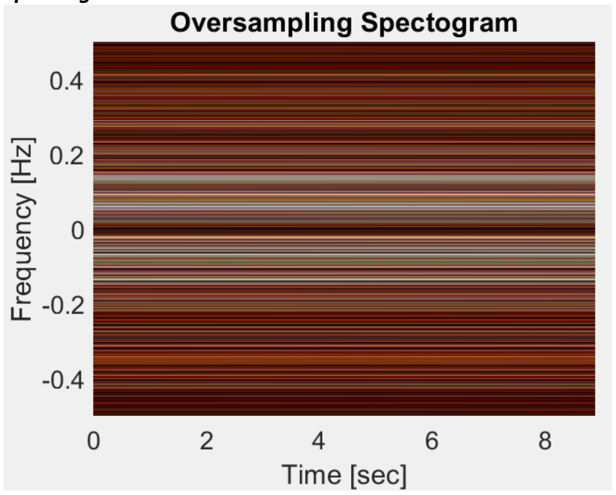
Spectrogram 2 – Mexican Hat Wavelet



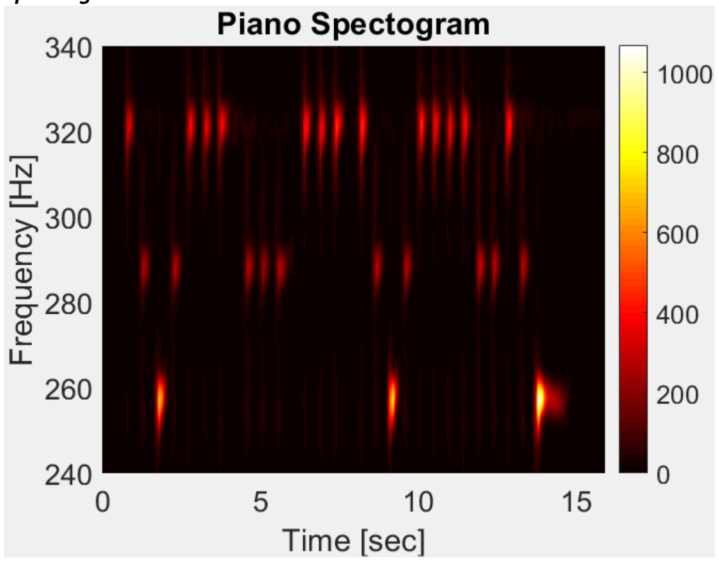
Spectrogram 3



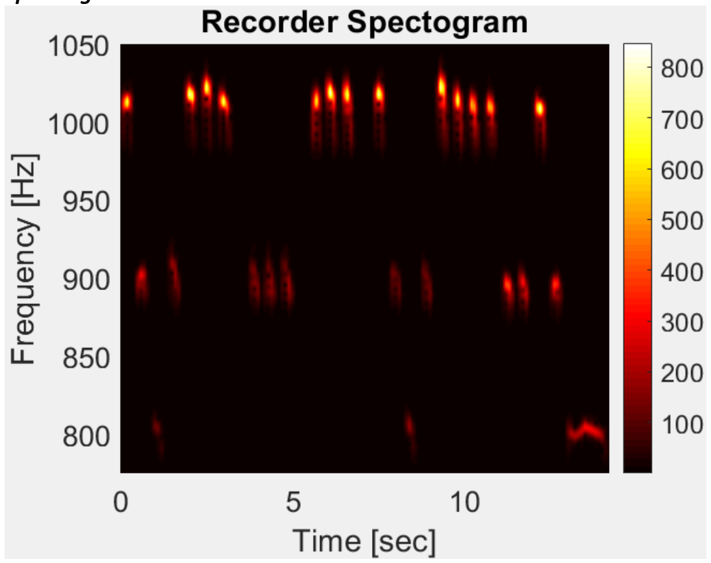
Spectrogram 4



Spectrogram 5 – PART 2: Piano



Spectrogram 6 – PART 2: Recorder



Part 2:

Spectrogram 5 - Part 2 Piano: From the spectrogram above to the left, we can find the frequency of which notes are played, and see their approximate duration from the spacing, and their intensity from the color bar. Bolded notes (-**Note**-) are played with higher intensity and played for slightly longer.

Order and relative spacing in time of notes on Piano:

E D C D EEE DDD EEE-**E**- D C D EEEE DDED -**C**-

Notes Played:

- (middle) C = 261.63 Hz.
- D = 293.66 Hz.
- E = 329.63 Hz.

Spectrogram 6 - Part 2 Recorder: From the recorder spectrogram above to the left, we can determine:

Approximate Notes Played:

- B = 987.77 Hz.
- A = 880.00 Hz.
- G =783.99 Hz.

Order and relative spacing in time of notes on recorder:

B A G A BBB AAA BBB **-B-** A G A BBBB AA B A -**G**-

To filter out the overtones, we know how wide to make the Gaussian Filter, and what frequency to center it at by looking at a non-filtered spectrogram and finding the center frequency of the range of notes. Then we set the gaussian filter width to enough to encompass all the notes, and get rid of all the overtones. This is how the Gaussian Filters for both the piano and the recorder samples were made.

Notice that the notes played on the recorder are at a much higher frequency than the notes played on the piano. This causes the overtones for the recorder to be much higher than the overtones for the piano. The overtones are what dictate the timbre of an instrument. Timbre is the perceived sound quality of a musical note, which is why a recorder sounds different than a piano. This is the main difference between a piano and a recorder, which can be clearly seen by looking at a spectrogram with no overtones filtered out.

Sec. V. Summary and Conclusions

From Part 1 we were able to see that different Gabor window widths changes the ratio of resolution in time and frequency. With a narrower Gabor window leading to higher time resolution and a wider Gabor window corresponding to higher frequency resolution. We also see that by oversampling we get extremely low time resolution, and very high frequency resolution, to the point of blurring the frequency resolution. By under sampling we can see that we have lost a lot of the frequency information from the signal data. Additionally, we were able to see how a Mexican Hat Wavelet changed the spectrogram.

In Part 2, we were able to determine the notes and relative duration of the notes for both the Piano and Recorder music samples. We were also able to find that the main difference between a piano and a recorder is the timbre, which can be seen in the spectrogram from the different overtones of the instruments.

Appendix A. MATLAB functions used and brief implementation explanation

fftn

- Y = fft(X).
- Computes the discrete Fourier transform of X using the Fast Fourier Transform Algorithm.
- *Implementation*: Used to convert the Gabor transformed signal from the time domain into the frequency domain

pcolor

- pcolor(X,Y,C).
- Creates a psucocolor plot using the values in the matrix C to specify the the colors. The size of C must match the size of the X, Y coordinate grid. Specifies the X and Y coordinates of the vertices.
- *Implementation*: Used to plot all spectrograms with time on the x axis and frequency on the y axis. Ex: Used in Part 1 (Line 109): pcolor(tslide,ks,Sgt_spec.').

Appendix B. MATLAB codes

Each part of the program (Part 1, Part 2: Piano, & Part 2: Recorder) was computed in a separate file.

Part 1 Code:

```
%% PART 1 Set Up - (lines 1-14)
clear; close all; clc
load handel

amplitude = y'; %amplitude of signal in the time domain
t = (1:length(amplitude))/Fs; %time vector in seconds
S = amplitude; %signal S equal to amplitude

%Defining frequencys and datapoints to eventually make the
spectogram
L=length(t);
n=length(amplitude); % n is the number of data points in signal
k=(1/L)*[0:(n-1)/2 -(n-1)/2:-1]; %Odd # of frequencys. Frequency
in Hz.
ks=fftshift(k); %shifts frequencys to go from negative values to
positive

%% Spectrograms for Varying Gabor Widths - (lines 15-46)
a_vec = [0.1 5 50 150]; % a_vec = different filter widths

figure(1)
%outer for loop rotates through the different filter widths
for jj = 1:length(a_vec)

    a = a_vec(jj); %sets new filter width a
    tslide=0:0.1:max(t); %slides the gabor window through the
signal in time
    Sgt_spec = zeros(length(tslide),n);

    %this computes the spectogram for each different filter
width a
    for j=1:length(tslide)

        g=exp(-a*(t-tslide(j)).^2); %Gabor Filter
        Sg=g.*S;
        Sgt=fft(Sg);
        Sgt_spec(j,:) = fftshift(abs(Sgt));

    end

    %Ploting
    subplot(2,2,jj)
    pcolor(tslide,ks,Sgt_spec.'),
    shading interp
    title(['width a = ',num2str(a)],'Fontsize',16);
    xlabel('Time [sec]');
    ylabel('Frequency [Hz]');
    colormap(hot)

end

%% Exploring Oversampling and Under Sampling - (lines 47-92)
%oversampling
figure(2)
a = 0.001; % this is a  wide filter
tslide=0:0.1:max(t); % slides the time vector
Sgt_spec = zeros(length(tslide),n); %re-defines the spectogram
freuqency vector

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

%ploting oversampling spectogram
pcolor(tslide,ks,Sgt_spec.'),
shading interp
title('Oversampling Spectogram');
xlabel('Time [sec]');
ylabel('Frequency [Hz]');
set(gca,'Fontsize',16)
colormap(hot)


%Undersampling
figure(3)
a = 1000; % this is a relly narrow
tslide=0:1:max(t); % time vector
Sgt_spec = zeros(length(tslide),n); %re-defines the spectogram
freuqency vector

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
```

```
%ploting under sampling spectogram
pcolor(tslide,ks,Sgt_spec.'),
shading interp
title('Under Sampling Spectogram');
xlabel('Time [sec]');
ylabel('Frequency [Hz]');
set(gca,'Fontsize',16)
colormap(hot)

%% Mexican Hat Wavelet - (lines 93-115)

figure(4)
a = 100; % this is a relatively narrow filter

tslide=0:0.1:max(t); % this makes the time vector "move barely"
Sgt_spec = zeros(length(tslide),n); % this re-defines the
spectogram freuqency vector

for j=1:length(tslide)
    mh = a*(1-(t-tslide(j)).^2).*exp(-((t-tslide(j)).^2)/2);
%mexican hat filter
    Sg = mh.*S;
    Sgt = fft(Sg);
    Sgt_spec(j,:) = fftshift(abs(Sgt));
end

%plot spectogram
pcolor(tslide,ks,Sgt_spec.'),
shading interp
title('Mexican Hat wavelet');
xlabel('Time [sec]');
ylabel('Frequency [Hz]');
set(gca,'Fontsize',16)
colormap(hot)
```

Part 2 Piano Code:

```
%% PART 2 (Piano) Set Up - (lines 1-14)
clear; close all; clc

%set up
[y,Fs] = audioread('music1.wav');
tr_piano=length(y)/Fs; % total recorded time in seconds
t = (1:length(y))/Fs; % vector of time in seconds
S = y'; %Signal S equal to the amplitude

L=tr_piano;
n=length(S); % n is the number of data points in signal
k=(1/L)*[0:n/2-1 -n/2:-1]; %EVEN # of frequencys in Hz
ks=fftshift(k);

%% Spectrogram - (lines 15-57)
tslide=0:0.1:tr_piano; %Descrete time vector
Sgt_spec = zeros(length(tslide),n);

%defining gaussian Filter
width = 0.00009;
centerF = 300;
filter = exp(-width*((ks - centerF).^2));

%gabor filter width
a = 150;

for j=1:length(tslide)

    %applying filter
    g=exp(-a*(t-tslide(j)).^2);
    Sg=g.*S;
    Sgt=fft(Sg);%fourier transforming to the freuqency domain

    %Gaussian filter function --> use non-shifted k because the
sgt uses
    %non-shifted k
    filter = exp(-width*((k - centerF).^2));

    Sgt = Sgt .* filter;

    %storing all the filtered signal at that time in the
spectogram vector
    Sgt_spec(j,:) = fftshift(abs(Sgt));

end
```

```
%plot spectogram
pcolor(tslide,ks,Sgt_spec.'),
shading interp
title('Piano Spectogram');
xlabel('Time [sec]');
ylabel('Frequency [Hz]');
set(gca,'Fontsize',16)
colormap(hot)
colorbar
ylim([240 340]);
p8 = audioplayer(y,Fs); playblocking(p8);
```

Part 2 Recorder Code:

```
%% Part 2 (Recorder) - (lines 1-20)
clear; close all; clc
[y,Fs] = audioread('music2.wav');
tr_rec=length(y)/Fs; % total recorded time in seconds
t = (1:length(y))/Fs; % vector of times
S = y'; %sets the signal equal to y (amplitude) of the filter

%setting frequency domain stuff
L=tr_rec;
n=length(S); % n is the number of data points in signal
k=(1/L)*[0:n/2-1 -n/2:-1]; %EVEN # of frequencys **** DEVIDE BY
L OR NAH???
% ^ by changing the scaling factor for k to (1/L) this gives the
freuqencys
% in Hz
ks=fftshift(k);

%gaussian Filter --> used to filter out overtones
width = 0.00005;
centerF = 900;
filter = exp(-width*((ks - centerF).^2));

%% Spectrogram - (lines 21-60)

tslide=0:0.1:tr_rec; % this makes the time vector in steps
Sgt_spec = zeros(length(tslide),n); % defines the spectogram
freuqency vector
%^ sgt_spec(time in increments of tslide, length of signal)

%gabor filter width
a = 150;

for j=1:length(tslide)

    %applying filter
    g=exp(-a*(t-tslide(j)).^2);
    Sg=g.*S;
    %fourier transforming to the freuqency domain
    Sgt=fft(Sg);

    %Gaussian filter function --> use non-shifted k because the
sgt uses
    %non-shifted k
    filter = exp(-width*((k - centerF).^2));

    Sgt = Sgt .* filter;
    %storing all the filtered signal at that time in the
spectogram vector
    Sgt_spec(j,:) = fftshift(abs(Sgt));
end

%plot spectogram
figure(2)
pcolor(tslide,ks,Sgt_spec.'),
shading interp
title('Recorder Spectogram');
xlabel('Time [sec]');
ylabel('Frequency [Hz]');
set(gca,'Fontsize',16)
colormap(hot)
colorbar
ylim([775 1050]);

p8 = audioplayer(y,Fs); playblocking(p8);
```