Music and Gabor

AMATH582: Homework 2
Maple Tan

February 15, 2019

1 Abstract

The first objective is to explore how different Gabor Transformations applied to Handel's *Hallelujah* affect the result of our spectrograms. We will observe how the Gabor Transformation will increase resolution in time in exchanged for frequency resolution or vice versa. Different types of filters will affect the resolution of time versus frequency as well. The second objective is to apply the Gabor Transformation to find the differences between a recorder and piano playing the same melody of *Mary Had a Little Lamb*. We see that despite playing the same notes, the flute plays at a higher frequency and does not have the same amount of overtones that the piano has.

2 Introduction and Overview

We start the first objective by loading Handel's song. We will essentially apply the Gabor Transformation to the song multiple times, changing an aspect of the transformation such as the translation, dilation and filter type, every single time. After graphing the spectrogram of each of these different filters, we will analyze the differences between them. For the second objective, we will use the same Gabor Transformation to extract the music score of *Mary Had a Little Lamb.* from our two different sound files and also try to determine the difference between the recorder and flute despite them playing the same notes.

3 Theoretical Background

The Gabor Transformation on a function f is defined as the following:

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

where g is a filter, usually a Guassian in the form:

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

Essentially the Gabor Transformation integrates over our function and then applies the Fourier Transformation to the function as it integrates. In a more practical sense, this transformation takes a signal or function and slices out a part of the signal from one time t_1 to another time t_2. We then apply the Fourier Transformation to that time slice and then shift the filter one time step forward to take a different time slice of the function. Again, we apply the Fourier Transformation to that time slice and then repeat all the way to the end of the function. The main difference between the Gabor Transformation and the regular Fourier Transformation is that we can now track how the frequencies change over time.

The Fourier Transformation returns all the different frequencies that make up a signal throughout all time but does not tell us when those frequencies might occurred. In our song example, while the Fourier Transformation can return all the notes in the song, it does not tell us when those notes occur. The Gabor Transformation on the other hand, includes that additional element of time and lets us know when those notes occur. However, what it trades in exchange for time is that the frequency resolution may not be as sharp compared to the Fourier Transformation. Ultimately, if we want more resolution in time within the Gabor Transformation, we will have to sacrifice resolution in frequency and vice versa.

4 Algorithm Implementation and Development

In part one, we will first load and graph Handel's Song to give us an idea as to what our signal looks like. Included with the data is a `Fs` variable which represents how many measurements of our song are made per second. Using this, we can then divide the length of our loaded song vector by `Fs` to the length of the song in seconds. Along with this, we also construct our Fourier Domain which must be shifted to correspond with the fact that the `fft` function shifts our domain and puts it on a 2π periodic.

We begin with a base case that we can use to compare our different Gabor Filters to. We start by using a simple Guassian Filter and then choose a reasonable width for our filter, say 1 for this example. We then create a time slide, which is a vector that splits our time variable into time steps where we will apply our filter. Using a for loop next, we then iterate through our time steps and apply the filter through the iterations. Then for each filtered signal, we apply the Fast Fourier Transform to it and then store the shifted version into a matrix that will store a snapshot of our function at that time slide.

After this base case has been created, we start varying the width of our base Guassian Filter. In this case, we increased the width by a factor of 10000 for the wider filter and decreased the width by that same factor for the smaller filter. For our second experiment, we then vary the translation, either taking more snapshots with a smaller time slide or taking less snapshots with a larger time slide. We increased and decreased the translation by a factor of 10 for the different cases. Finally, we vary the types of filter we use. We repeat the base case with a less square looking Guassian Filter, a Shannon Filter and a Mexican Hat Filter.

For part two, we begin by reading the two .wav files using the `audioread` function in MATLAB. This function reads an audio file and returns a vector that contains the signal data in Hertz. Due to memory limitations on the working device, the length of the two songs had to be cut in half but the process is still the same and the code to achieve the same result with the full length song has been included in the appendix. The shorten version of the vectors will cover the first four bars of the song in both versions. One thing important to note is that depending whether our signal vector `n` is of even length or odd length, our associated Fourier Domain must be adjusted accordingly, `(2*pi)*[0:n/2 -n/2:-1]` if even and `(2*pi)*[0:n/2-1 -n/2:-1]` if odd.

We then apply the same Gabor Transformation to both signals in a similar fashion to part 1 and graph the associated spectrogram. In order to see the differences between the recorder and piano playing this song, we keep the overtones shown in the spectrogram. We also limit the y axis to positive values to match what a music score appearance. Since the negative frequencies are the same as their positive counterpart just at the bottom of their wave instead top.

5 Summary and Conclusion

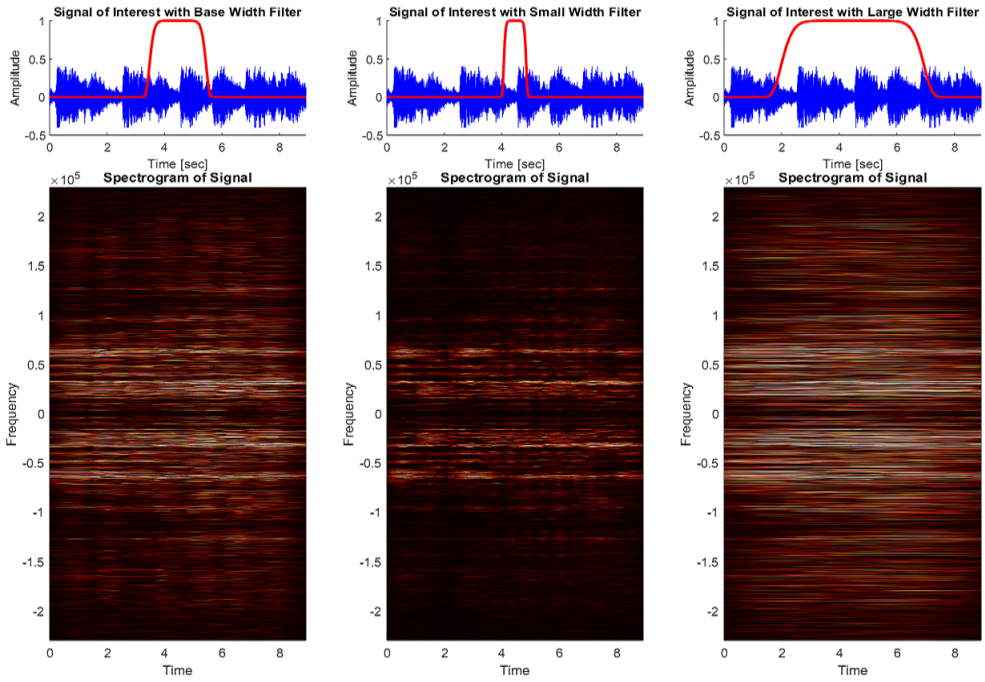

Figure 1: Three different spectrograms that show a visualization of how filter width affects the resolution of our time versus frequency axis.

Beginning with part 1, we see that changing the filter width resulted in the following spectrograms seen in Figure 1 above. While a larger filter width gives us a clearer picture on all frequencies that occur during the song, we completely lose resolution in time. On the other hand, making the filter width smaller gives us a much clearer idea as to when the different frequencies are happening but then we lose visual on some of the weaker frequencies that can be seen in the base and large filter width cases.

Then when we adjust the translation, we see that the smoothness of the spectrogram changes. With the larger time slide, we see that the spectrogram is a lot more jagged compared to the spectrograms with smaller time slides, though the general shape in terms of time versus frequency resolution did not change very much between the different trials. One thing to note as well is that there is not a very large improvement between the base time slide and the small time slide, but the small time slide took a lot longer when it came to run time.

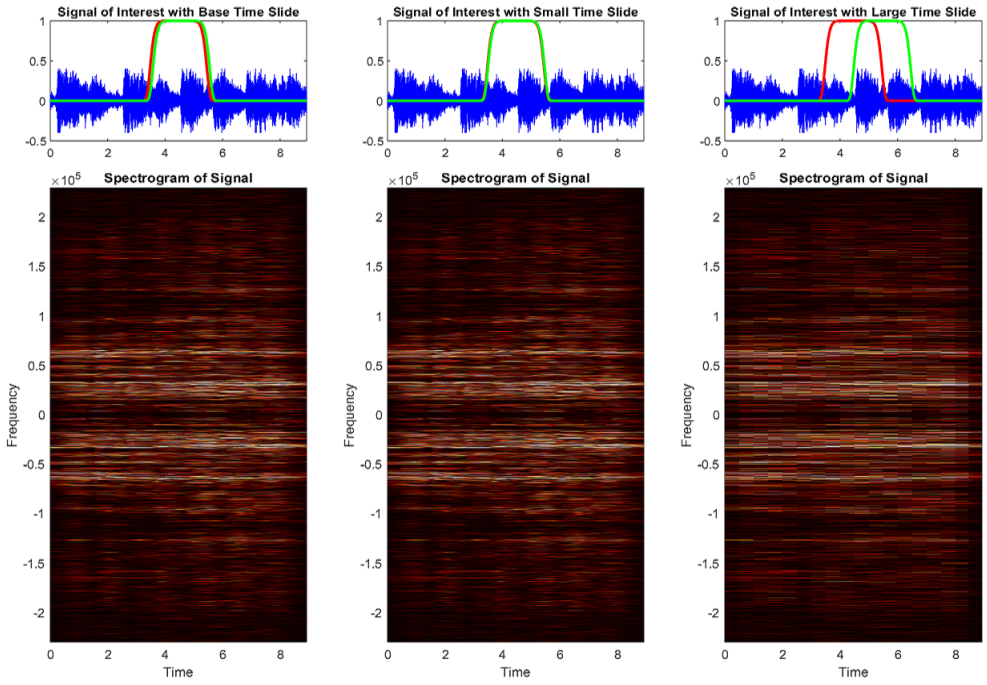

Figure 2: Three different spectrograms that visualizes how the size of the time slide changes the resulting spectrogram.

As for changing the filter type, we see that in Figure 3 that the less square Guassian filter completely removes the time element but the frequency resolution is very clear. Using the Shannon Filter produces a similar result to our base Guassian Filter, most likely due to their shapes being fairly similar. We can see that some edges look sharper which could be attributed to the Shannon Filter's sharp edges. Then with the Mexican Hat Filter, we see more resolution in the time axis and it looks more similar to our base Guassian with a large width.

Finally for our second objective, we see that the main difference between the music scores is that the recorder plays notes at a higher frequency than the piano. So even though they are technically playing the same note, the recorder is playing the note maybe one or two octaves higher. Another noted difference that the piano generates many more overtones compared to the recorder. For example, we can see that the first note in the piano music score very clearly around $\omega =\sim 1.6 \times 10^4$ Hz and we can also see more fainter signals at $\sim 3.2 \times 10^4$ Hz and $\sim 4.8 \times 10^4$ Hz as well, which are 2ω and 3ω respectively. We do not see these types of overtones in the recorder sheet music.

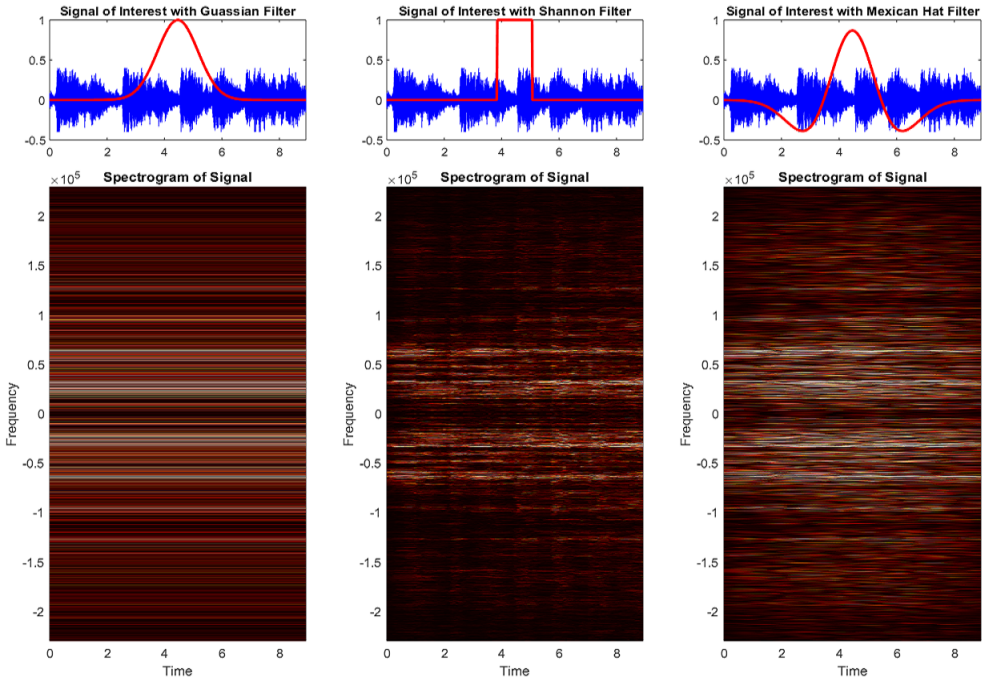

Figure 3: Figure showing how different filter types affect the resulted spectrograms. We see that despite having similar base dilation, the resulting resolution of frequency versus time can vary immensely between filter types.

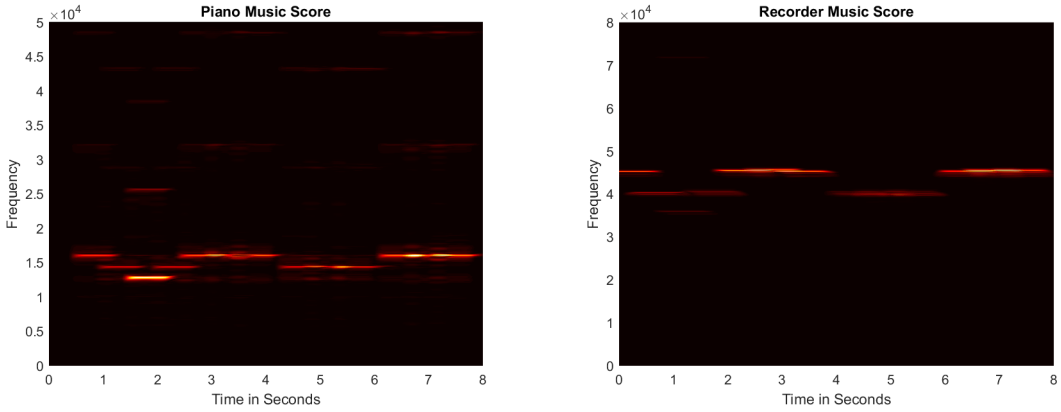

Figure 4: We see the generated Piano Music Score on the right and the generated Recorder Music Score on the left for *Mary had a Little Lamb*. Note the difference in y scales, the recorder plays notes at a higher frequency than the piano.

A MATLAB CODE - Part 1

```
clear all; close all; clc

load handel

%% Setting up the Data
v = y'/2; % The music data
n = length(v);  % Length of time t
t = (1:length(v))/Fs;  % Time t in seconds
L = max(t); $ End time
k = (2*pi)*[0:n/2 -n/2:-1]; % Fourier Domain


%% Try Different Filter Window Widths
tslide = 0:0.1:10;
spc1 = [];
spc2 = [];
spc3 = [];
width = [1, 10000, 0.0001];
for j=1:length(tslide)
    g1=exp(-width(1)*(t-tslide(j)).^10);
    g2=exp(-width(2)*(t-tslide(j)).^10);
    g3=exp(-width(3)*(t-tslide(j)).^10);
    vf1 = g1.*v;
    vft1 = fft(vf1);
    vf2 = g2.*v;
    vft2 = fft(vf2);
    vf3 = g3.*v;
    vft3 = fft(vf3);
    spc1 = [spc1; abs(fftshift(vft1))];
    spc2 = [spc2; abs(fftshift(vft2))];
    spc3 = [spc3; abs(fftshift(vft3))];
end

midpoint = max(t) / 2;
g1=exp(-width(1)*((t-midpoint).^10));
g2=exp(-width(2)*((t-midpoint).^10));
g3=exp(-width(3)*((t-midpoint).^10));
```

```
%% Plotting Top Part of Figure
clf
set(gcf, 'Position',  [100, 100, 1200, 800])
subplot(4,3,1)
hold on
plot(t,v, 'b')
plot(t, g1, 'r','linewidth',2)
title('Signal of Interest with Base Width Filter');
xlabel('Time [sec]');
xlim([0 t(end)]);
ylabel('Amplitude');

subplot(4,3,2)
hold on
plot(t,v, 'b')
plot(t, g2, 'r','linewidth',2)
title('Signal of Interest with Small Width Filter');
xlabel('Time [sec]');
ylabel('Amplitude');
xlim([0 t(end)]);

subplot(4,3,3)
hold on
plot(t,v, 'b')
plot(t, g3, 'r','linewidth',2)
title('Signal of Interest with Large Width Filter');
xlabel('Time [sec]');
ylabel('Amplitude');
xlim([0 t(end)]);

%% Plotting Bottom Part of Figure

subplot(4,3, [4 7 10])
pcolor(tslide, fftshift(k), spc1.'), shading interp, colormap(hot)
title('Spectrogram of Signal')
xlim([0 t(end)]);
xlabel('Time')
ylabel('Frequency')

subplot(4,3, [5 8 11])
```

```
pcolor(tslide, fftshift(k), spc2.'), shading interp, colormap(hot)
title('Spectrogram of Signal')
xlim([0 t(end)]);
xlabel('Time')
ylabel('Frequency')

subplot(4,3, [6 9 12])
pcolor(tslide, fftshift(k), spc3.'), shading interp, colormap(hot)
title('Spectrogram of Signal')
xlim([0 t(end)]);
xlabel('Time')
ylabel('Frequency')

%% Try Different Time Slides
midpoint = max(t) / 2;
width = 1;
g=exp(-width(1)*((t-midpoint).^10));

%% Step = 0.1
step1 = 0.1;
tslide1 = 0:step1:10;
spc1 = [];
width = 1;
for j=1:length(tslide1)
    g1=exp(-width*(t-tslide1(j)).^10);
    vf = g1.*v;
    vft = fft(vf);
    spc1 = [spc1; abs(fftshift(vft))];
end

%% Step2 = 1
step2 = 1;
tslide2 = 0:step2:10;
spc2 = [];
width = 1;
for j=1:length(tslide2)
    g2=exp(-width*(t-tslide2(j)).^10);
    vf = g2.*v;
    vft = fft(vf);
    spc2 = [spc2; abs(fftshift(vft))];
```

```
end

%% Step3 = 0.01
step3 = 0.02;
tslide3 = 0:step3:10;
spc3 = [];
for j=1:length(tslide3)
    g3=exp(-width*(t-tslide3(j)).^10);
    vf = g3.*v;
    vft = fft(vf);
    spc3 = [spc3; abs(fftshift(vft))];
end

%% Plotting Top Part of Figure
g1=exp(-width*((t-midpoint-step1).^10));
g2=exp(-width*((t-midpoint-step2).^10));
g3=exp(-width*((t-midpoint-step3).^10));
clf;
set(gcf, 'Position',  [100, 100, 1200, 800])
subplot(4,3,1)
plot(t,v, 'b')
hold on
plot(t, g, 'r','linewidth',2)
plot(t, g1, 'g','linewidth',2)
xlim([0, t(end)])
title('Signal of Interest with Base Time Slide');

subplot(4,3,2)
plot(t,v, 'b')
hold on
plot(t, g, 'r','linewidth',2)
plot(t, g3, 'g','linewidth',2)
xlim([0, t(end)])
title('Signal of Interest with Small Time Slide');

subplot(4,3,3)
plot(t,v, 'b')
hold on
plot(t, g, 'r','linewidth',2)
plot(t, g2, 'g','linewidth',2)
```

```
title('Signal of Interest with Large Time Slide');
xlim([0, t(end)])


%% Plotting Bottom Part of Figure
subplot(4,3, [4 7 10])
pcolor(tslide1, fftshift(k), spc1.'), shading interp, colormap(hot)
title('Spectrogram of Signal')
xlim([0, t(end)])
xlabel('Time')
ylabel('Frequency')


subplot(4,3, [5 8 11])
pcolor(tslide3, fftshift(k), spc3.'), shading interp, colormap(hot)
title('Spectrogram of Signal')
xlim([0, t(end)])
xlabel('Time')
ylabel('Frequency')


subplot(4,3, [6 9 12])
pcolor(tslide2, fftshift(k), spc2.'), shading interp, colormap(hot)
title('Spectrogram of Signal')
xlim([0, t(end)])
xlabel('Time')
ylabel('Frequency')


%% Try Different Filter Types

% Guassian Filter
midpoint = max(t) / 2;
width = 1;
g=exp(-width(1)*((t-midpoint).^2));

% Shannon Filter
Fs = zeros(1,n);
midind = round(n / 2);
width_sh = 5000;
```

```
Fs(midind-width_sh:1:midind+width_sh)=ones(1, 2*width_sh+1);

% Mexican Hat Filter
sigma = 1;
m = (2/(sqrt(3*sigma)*pi^(1/4))) ...
.*((1-((t-midpoint)/sigma).^2) ...
.*exp(-((t-midpoint).^2)/ (2*sigma^2)));

% Set up Starting and Ending Positions of Shannon Filter
tslide = 0:0.1:10;
indjump = ones(1, 101);
for j=1:length(tslide)-1
    indjump(j+1) = indjump(j) + 725;
end

% Looping through Time Slides
spc_guass = [];
spc_shan = [];
spc_mehat = [];
for j=1:length(tslide)
    ga=exp(-width*(t-tslide(j)).^10);
    mehat = (2/(sqrt(3*sigma)*pi^(1/4))) ...
     .*((1-((t-tslide(j))/sigma).^2) ...
     .*exp(-((t-tslide(j)).^2)/(2*sigma^2)));

    sh = zeros(1,n);
    sh_start = indjump(j)-width_sh;
    if sh_start < 1
        sh_start = 1;
    end
    sh_end = indjump(j)+width_sh;
    if sh_end > n
        sh_end = n;
    end
    current_shannon_width = sh_end-sh_start;
    sh(sh_start:1:sh_end)=ones(1, current_shannon_width+1);

    vfg = g.*v;
    vftg = fft(vfg);
    spc_guass = [spc_guass; abs(fftshift(vftg))];
```

```
    vfsh = sh.*v;
    vftsh = fft(vfsh);
    spc_shan = [spc_shan; abs(fftshift(vftsh))];

    vfm = mehat.*v;
    vftm = fft(vfm);
    spc_mehat = [spc_mehat; abs(fftshift(vftm))];
end


%% Plot Top Part of Figure
set(gcf, 'Position',  [100, 100, 1200, 800])
clf
subplot(4,3,1)
plot(t,v, 'b')
hold on
plot(t, g, 'r','linewidth',2)
xlim([0, t(end)])
title('Signal of Interest with Guassian Filter');

subplot(4,3,2)
plot(t,v, 'b')
hold on
plot(t,v, 'b');
plot(t, Fs, 'r', 'linewidth', 2);
xlim([0, t(end)])
title('Signal of Interest with Shannon Filter');

subplot(4,3,3)
plot(t,v, 'b')
hold on
plot(t, m, 'r','linewidth',2)
title('Signal of Interest with Mexican Hat Filter');
xlim([0, t(end)])

%% Plot Bottom Part of Figure

subplot(4,3, [4 7 10])
pcolor(tslide, fftshift(k), spc_guass.'), shading interp, colormap(hot)
```

```
title('Spectrogram of Signal')
xlim([0 t(end)]);
xlabel('Time')
ylabel('Frequency')

subplot(4,3, [5 8 11])
pcolor(tslide, fftshift(k), spc_shan.'), shading interp, colormap(hot)
title('Spectrogram of Signal')
xlim([0 t(end)]);
xlabel('Time')
ylabel('Frequency')

subplot(4,3, [6 9 12])
pcolor(tslide, fftshift(k), spc_mehat.'), shading interp, colormap(hot)
title('Spectrogram of Signal')
xlim([0 t(end)]);
xlabel('Time')
ylabel('Frequency')
```

B MATLAB CODE - Part 2 - Memory Insufficient Version

```
set(gcf,'visible','off')

% Due to memory limitations, we will only look at the first four bars of the song
tr_piano= 16 / 2; % total record time in seconds divided in half

py=audioread('music1.wav');
py = py';

% Take seconds 0-8 of the song
pstart = 1;
pstop = (length(py)/2);
py = py(1:pstop);

pn = length(py);
pk = (2*pi)*[0:pn/2-1 -pn/2:-1];

%%
```

```
timeslide=0:0.05:tr_piano;
pt = linspace(0,8,pn);
p_spc = zeros(length(timeslide), length(py));

%%
width = 10000;
for n=1:length(timeslide)
    g = exp(-width*(pt-timeslide(n)).^10);
    pyf = py.*g;
    pyft = fft(pyf);
    p_spc(n,:) = abs(fftshift(pyft));
end

%%
figure(2)
set(gcf,'visible','off')
clf;
pcolor(timeslide, fftshift(pk), p_spc.'), shading interp, colormap(hot)
ybounds = 5*10^4;
ylim([0 ybounds])
title('Piano Music Score')
xlabel('Time in Seconds')
ylabel('Frequency')

print(gcf,'-dpng','pspect.png')

%% Take First Half of Flute Song
clear; clc

tr_rec=8;
ry=audioread('music2.wav');
ry = ry';
ry_end = length(ry)-8;
ry = ry(1:ry_end);
rstart = (length(ry)/14)*0+1;
rstop = (length(ry)/14)*7;
ry=ry(rstart:rstop);

rn = length(ry);
rk = (2*pi)*[0:rn/2-1 -rn/2:-1];
```

```
%% Plot Original Piano Playing Mary Had %% Little Lamb Signal
set(gcf,'visible','off')
Fs=length(ry)/tr_rec;
plot((rstart:rstop)/Fs,ry);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Mary had a little lamb (recorder)');
print(gcf, '-dpng','mary_rec.png')

%% Create Time Slide
timeslide=0:0.05:tr_rec;
rt = linspace(0,8,rn);
spc_r = zeros(length(timeslide), length(rt));

%% Iterate Through Time Slides
width = 1000;
for n=1:length(timeslide)
    g = exp(-width*(rt-timeslide(n)).^10);
    ryf = ry.*g;
    ryft = fft(ryf);
    spc_r(n,:) = abs(fftshift(ryft));
end

%% Plot Recorder Playing Mary Had a
%%  Little Lamb Signal
figure(2)
clf;
set(gcf,'visible','off')
pcolor(timeslide, fftshift(rk), spc_r.'), shading interp, colormap(hot)
ybounds = 8*10^4;
ylim([0 ybounds])
title('Recorder Music Score')
xlabel('Time in Seconds')
ylabel('Frequency')
```

C MATLAB CODE - Part 2 - Memory Sufficient Version

```
py=audioread('music1.wav');
py = py';
pn = length(py);
pk = (2*pi)*[0:pn/2-1 -pn/2:-1];

%%
timeslide=0:0.05:tr_piano;
pt = linspace(0,16,pn);
p_spc = zeros(length(timeslide), length(py));

%%
width = 10000;
for n=1:length(timeslide)
    g = exp(-width*(pt-timeslide(n)).^10);
    pyf = py.*g;
    pyft = fft(pyf);
    p_spc(n,:) = abs(fftshift(pyft));
end

%%
figure(2)
set(gcf,'visible','off')
clf;
pcolor(timeslide, fftshift(pk), p_spc.'), shading interp, colormap(hot)
ybounds = 5*10^4;
ylim([0 ybounds])
title('Piano Music Score')
xlabel('Time in Seconds')
ylabel('Frequency')

print(gcf,'-dpng','pspect.png')

%%
clear; clc
tr_rec=14;
ry=audioread('music2.wav');
```

```
ry = ry';
rn = length(ry);
rk = (2*pi)*[0:rn/2-1 -rn/2:-1];

%%
set(gcf,'visible','off')
Fs=length(ry)/tr_rec;
plot((rstart:rstop)/Fs,ry);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Mary had a little lamb (recorder)');
print(gcf, '-dpng','mary_rec.png')

%%
timeslide=0:0.05:tr_rec;
rt = linspace(0,14,rn);
spc_r = zeros(length(timeslide), length(rt));

%%
width = 1000;
for n=1:length(timeslide)
    g = exp(-width*(rt-timeslide(n)).^10);
    ryf = ry.*g;
    ryft = fft(ryf);
    spc_r(n,:) = abs(fftshift(ryft));
end

%%
figure(2)
clf;
set(gcf,'visible','off')
pcolor(timeslide, fftshift(rk), spc_r.'), shading interp, colormap(hot)
ybounds = 8*10^4;
ylim([0 ybounds])
title('Recorder Music Score')
xlabel('Time in Seconds')
ylabel('Frequency')
print(gcf, '-dpng','rspect.png')
```