Amath 482 homework 2: Gabor Transform

Andy Zhang

February 8, 2020

Abstract

This report demonstrates the applications of Gabor Transform, a way to discrete the original data into windows that hold information in both spatial / time and frequency domains. Specifically, we're going to focus on the time-frequency analysis of music pieces Handel's Messiah and Mary had a little lamb.

1 Introduction

The time-frequency analysis in this report is based on Gabor Transform, which allows us to demonstrate information in both frequency and time domains.

In the first part of this report, we're going to explore on how different parameters, including the window width of Gabor Transform, translation of windows in time domain, and three commonly-used wavelet functions for Gabor Transform, change the resolution of time and frequency data of a music piece of Handel's Messiah.

In part two, a Gaussian function is applied to music pieces of Mary had a little lamb played by piano and recorder to do time-frequency analysis so that we can determine the different frequencies and music scores played by different musical instruments.

2 Theoretical Background

2.1 Gabor Transform

Since time-series analysis and Fourier Transform can only focus in a specific domain, such limitations brought into the idea of Gabor Transform, a modification that can extract both time and frequency information. It's defined as:

$$\mathcal{G}[f](t,\omega) = \tilde{f}_g(t,\omega) = \int_{-\infty}^{\infty} f(\tau)\bar{g}(\tau - t)e^{-i\omega\tau}d\tau = (f, \bar{g}_{t,\omega}) \tag{1}$$

where the bar denotes the complex conjugate of the function, thus, the function $g(\tau - t)$ acts as a window of data at each instant time and information outside

the window are filtered out. The integration over the parameter τ slides this window over the time domain so that the frequency information is extracted.

We always face the trade-off between resolutions of time and frequency. Increasing the width of window results in a better frequency resolution and a worse time resolution; narrowing the window does the opposite.

2.2 Wavelet

The idea of wavelets is introduced to gain perfect time and frequency resolutions by modifying the window size, that is, we can extract low-frequency components by using a wide scaling window and through successively narrowing of our window, higher-frequency details and better time-resolution are extracted. The mother wavelet function is defined as:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi(\frac{t-b}{a}) \tag{2}$$

where $a \neq 0$ and b are real constants. The scaling parameter a controls the successively changing window width and translation parameter b moves our window along the signal.

In Part 1, three commonly-used wavelets are applied, which are:

Gaussian window The simplest wavelet among the three, it's defined as:

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

with window width of a (the size of a is directly proportional to the time resolution due to narrower window width) and the centered time τ along the signal.

Mexican hat window The Mexican hat window is the negative normalized second derivative of a Gaussian function. It's more commonly used due to its excellent localization properties in both time and frequency space. The function of Mexican hat window is applied as:

$$\psi(t) = \frac{2}{\sqrt{3\sigma}\pi^{1/4}}(1-(\frac{t-\tau}{\sigma})^2)e^{(-\frac{(t-\tau)^2}{2\sigma^2})} \tag{4}$$

where τ is still the centered time and σ being the window width. As you may notice, σ is always at the denominator, as a result, a small σ creates a narrow window and a better resolution in time domain.

Shannon window The idea of Shannon window is that, it eliminates all information outside the window and keeps what's inside the same. To accomplish that, we can use a gate function:

$$\psi(t) = \begin{cases} 0 & \text{if } |t-\tau| \geq \text{a} \\ 1 & \text{if } |t-\tau| < \text{a} \end{cases} \tag{5}$$

that creates a window width of $2a$ around centered time τ. The size of a is directly related with window width and better time resolution.

2.3 Spectrogram

The spectrogram is the visual representation of the spectrum of frequencies along the signal verses time. We can intuitively view time-frequency information and do analysis with the help of spectrogram.

3 Algorithm Implementation and Development

Part I

Prepare for Gabor Transform

To start with, I load the music piece and get its sample rate and amplitude information with 73113 nodes. Due to the periodic boundary conditions, I remove the last amplitude detail. After dividing the 73112 nodes by the sample rate, I round the result to 9 seconds and name it **L**. Then I rescale the frequencies **k** by $2\pi/L$ because Fast Fourier Transform assumes 2π periodic signals and the domain ranges from 0 to L, that is a total L units long. I also **fftshift()** the frequencies so that the zero frequency is staying at the middle of the array.

Explore on the parameter of window width, translation of time, and different window types

I create an array **tslide** from 0 to **L**, with a step of 0.1 because this step is appropriate for the total length of **L**. Then I test on four different widths of 1, 5, 25, 50. Since we're dealing with Gaussian window in this part, so the larger of width parameter, the better resolution we get about time and worse about frequency. Then I substitute each width into the function and multiply it with the amplitude data **v** and do Fourier Transform, the result is called **vgt**. I build a matrix called *vgtspec* that holds information of the absolute value of fft-shifted **vgt** at every time step. Next, I can create a spectrogram for each of the width above by calling $pcolor(tslide, ks, vgtspec.')$, where ks is the shifted frequency and $vgtspec.'$ denotes the transpose of the matrix. These spectrograms are shown below in **Computational Results**.

Applying the same idea as above, I set the width to be 25 since it produces the spectrogram that demonstrates both time and frequency information best among the four. Then I create different **tslides**, all ranging from 0 to **L**, with different time steps of 0.05, 0.1, 0.5, and 1.5. Applying the same method as above, I create another four spectrograms. Different **tslides** cause the problems of oversampling (too small time step) and undersampling (too large time step), which will be further discussed in **Computational Results**.

I apply the ideal width of 25 and ideal translation period of 0.1 second in this scenario for Gaussian window. By applying the functions of Mexican hat window and Shannon window to the amplitude data, I create their respective

spectrograms after several attempts to find the ideal parameters. I also plot each window with the amplitude plot so that we can intuitively view their shapes and understand how they filter out data.

Part II

Prepare for Gabor Transform

To start with, I load the music pieces of Mary had a little lamb played by piano and recorder. Applying similar idea as the preparation for **Part one** except removing the last amplitude detail because the data don't show periodic boundary condition. I also round the length of piano recoding to 16 seconds (exact duration about 15.9 seconds) and that of recorder to 14.2 seconds. Note that since we're dealing with sound data, we need the frequencies in unit of hertz, so we have to rescale the frequencies **k** by $1/L$ instead of $2\pi/L$ because that will end up with frequencies in unit of radian.

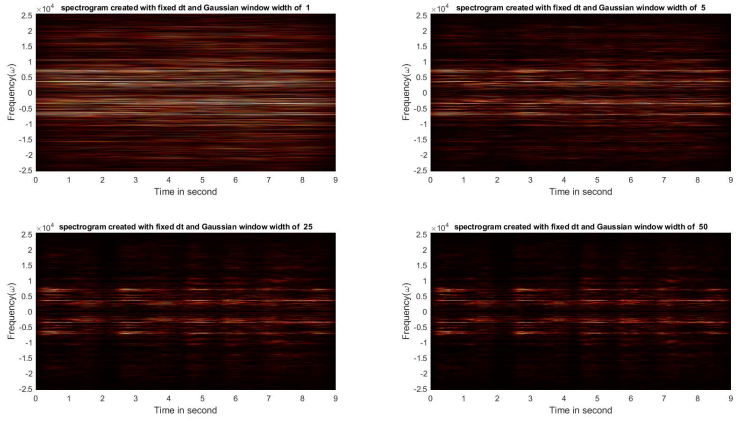

Figure 1: Spectrograms generated by Gaussian window with fixed time step and different width

Frequency plot, spectrogram, and identify music notes

Using the same method described in **Part one**, I create a spectrogram for music pieces played by each of the instruments by applying a Gaussian window with width parameter of 30 and a time step of 0.1. Additionally, I find the **index** of the maximum element in **vgt** at each time step, similar to the method applied in **An Ultrasound Problem**, so that I can find the center frequency at each time step in **k** with corresponding index. By doing this, we can filter out

the overtone and get a clean graph of frequencies. A matrix named **frequency** is created to store the center frequency along **L**.

As you may notice, the signal shown in spectrogram is intermittent and that can tell us the total number of notes being played or the duration of notes. By corresponding each separated note in spectrogram with the frequencies shown in frequency plot, we can identify the notes being played with the help of a musical scale.

4 Computational Results

Part 1

Figure 1 shows how different width parameters affect the spectrogram. On top left when $a = 1$, we can hardly tell any change of frequency in time space; while the image on bottom right gives perfect resolution in time but almost no changes in frequency ($a = 50$). The image on bottom left seems to be a good one since it shows good resolution of time and we can notice changes in frequency space. So 25 seems to be a good choice of Gaussian window width.

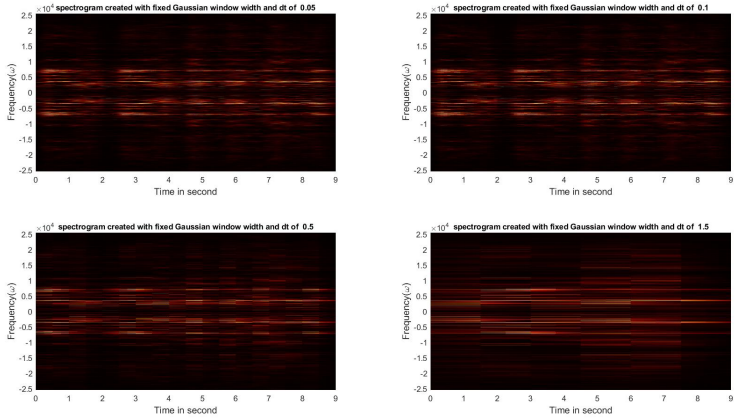

Figure 2: Spectrograms generated by Gaussian window with fixed width and different time step

Figure 2 shows how spectrogram is changed with same width of 25 and different time steps. While t step is too small like 0.05, we face "oversampling", that is, we've already had a good spectrogram so that we don't need to further decrease t step. If t step is too small, it's inefficient to go through loops. On the other hand, if t step is too large like 1.5, we can notice that we've got bad resolutions in frequency and time spaces because the step size is far larger than the window size and thus we lose data. And this is the problem of "undersampling". Among the four images, having the translation step as 0.1 is a good

choice.

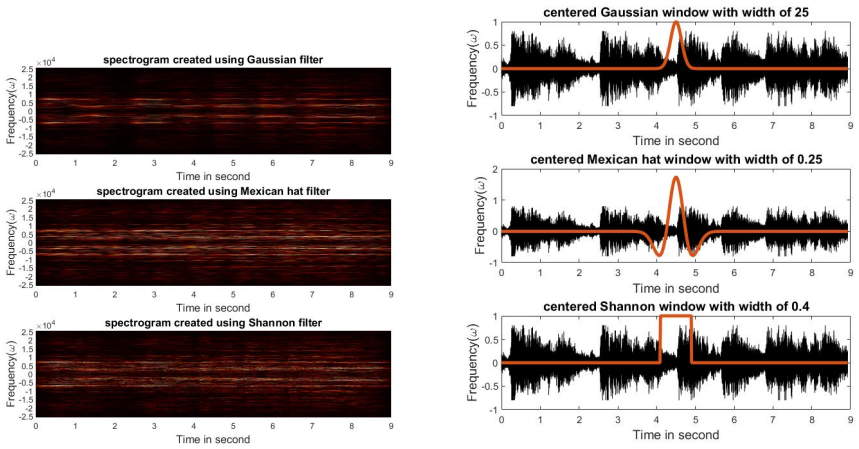

Figure 3: Spectrograms generated by different windows

On the left of Figure 3 demonstrates how different windows generate different spectrograms. With the choice of 0.25 as width and a translation period of 0.1, Mexican hat window creates much better frequency resolution than applying Gaussian window. The Shannon window shows better resolution in time space.

On the right shows how these windows differ in shapes when being applied to the data.

Part 2

The case of piano

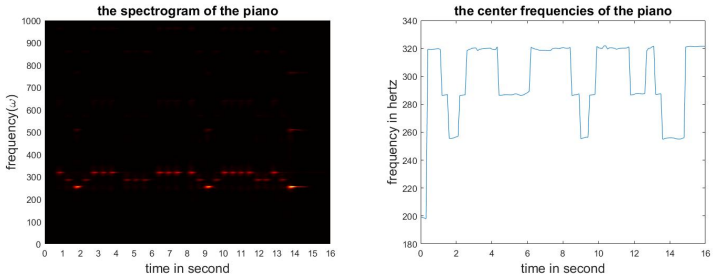

Figure 4: Spectrogram and frequency plot of the piano

On the left of Figure 4, the spectrogram shows that there are totally 26 notes being played and we can then find their corresponding frequencies on the

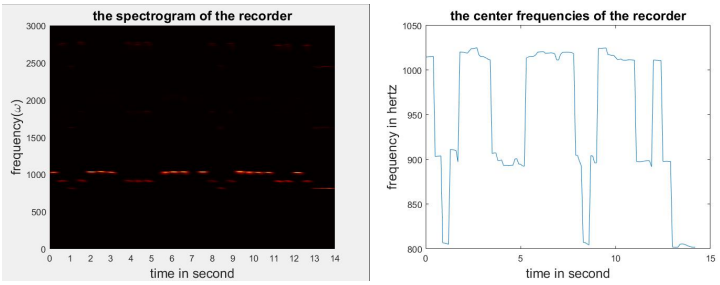

Figure 5: Spectrogram and frequency plot of the recorder

frequency plot. Thus, we know the music scores are: **E D C D E E E D D D E E E E D C D E E E E D D E D C**.

The case of recorder

As the spectrogram of Figure 5 shows, there are also 26 notes being played and through referring to the frequency plot, we know the music scores are approximately: **B A G A B B B A A A B B B B A G A B B B B A A B A G** since the frequencies are a little higher than these scores.

5 Summary and Conclusion

To sum up this report, we should know how different parameters, including window width, translation period, and window types, change the result of the spectrogram. We should know to choose proper window types and are able to find ideal values of parameters in all kinds of situations. Besides, we should know to filter out overtones by finding central frequencies and read from spectrograms. One should be capable to do time-frequency analysis after reading this report.

6 Appendix A

vt = fft(v) returns the Fourier Transform of a vector v.

ks = fftshift(k) rearranges the frequency data by shifting the zero frequency component to the middle of the array.

y = audioread('music.wav') reads data from the file named music.wav.

pcolor(tslide, ks, vgt spec creates a pseudocolor plot of matrix vgt spec on the grid defined by tslide and ks.

colormap(hot) sets the current figure's colormap to hot.

shading interp sets the current shading to interpolated.

7 Appendix B

```
clear all; close all; clc;

% Part 1

load handel
v = y';
%{
figure(1)
plot((1:length(v))/Fs,v);
xlabel('Time [sec]');
ylabel('Amplitude');
title('Signal of Interest, v(n)');
hold off
%}

% prepare data
L = 9;
v = v(1 : length(v) - 1);
n = length(v);
t = (1:length(v)) / Fs;
k= (2*pi/L)*[0:(n/2-1), -n/2:-1];
ks = fftshift(k);

% create Gabor Filters

% explore on the effect of different gabor window width (
    delta t of 0.1 is
% applied here, set to be the controlled variable)

width = [1, 5, 25, 50];
tslide = 0 : 0.1: L;

for i = 1 : 4
    vgt_spec = [];
    for j = 1 : length(tslide)
        gabor = exp(-width(i) * (t - tslide(j)).^2);
        vg = gabor.* v;
        vgt = fft(vg);
        vgt_spec = [vgt_spec; abs(fftshift(vgt))];
    end
    figure(2)
    subplot(2, 2, i)
    pcolor(tslide, ks, vgt_spec.')
```

```
43      shading interp
44      colormap(hot)
45      xlabel('Time in second', 'FontSize', 12)
46      ylabel('Frequency(\omega)', 'FontSize', 12)
47      title(['spectrogram created with fixed dt and
            Gaussian window width of  ', num2str(width(i))], '
            FontSize', 10)
48      xticks([0:1:L])
49      yticks([-25000:5000:25000])
50  end
51
52
53  % explore on the idea of oversampling and undersampling,
        that is, applying
54  % different translations of gabor window (window width is
        fixed in this
55  % time as a controlled variable, an appropriate width of
        25 is applied)
56
57
58  width = 25;
59  tslide = [];
60  dt = [0.05, 0.1, 0.5, 1.5];
61  tslide{1} = 0:0.05:L;
62  tslide{2} = 0:0.1:L;
63  tslide{3} = 0:0.5:L;
64  tslide{4} = 0:1.5:L;
65  for i = 1 : 4
66      vgt_spec = [];
67      for j = 1 : length(tslide{i})
68          gabor = exp(-width * (t - tslide{i}(j)).^2);
69          vg = gabor.* v;
70          vgt = fft(vg);
71          vgt_spec = [vgt_spec; abs(fftshift(vgt))];
72      end
73      figure(3)
74      subplot(2, 2, i)
75      pcolor(tslide{i}, ks, vgt_spec.')
76      shading interp
77      colormap(hot)
78      xlabel('Time in second', 'FontSize', 12)
79      ylabel('Frequency(\omega)', 'FontSize', 12)
80      title(['spectrogram created with fixed Gaussian
            window width and dt of  ', num2str(tslide{i}(2) -
            tslide{i}(1))], 'FontSize', 10)
81      xticks([0:1:L])
```

```
    yticks([-25000:5000:25000])
end



% explore on different gabor windows(Gaussian, Mexican hat, and Shannon)

% Gaussian
figure(4)
vgt_spec = [];
tslide = 0 : 0.1 : L;
for j = 1 : length(tslide)
    gabor = exp(-25 * (t - tslide(j)).^2);
    vg = gabor.* v;
    vgt = fft(vg);
    vgt_spec = [vgt_spec; abs(fftshift(vgt))];
end
subplot(3, 1, 1)
pcolor(tslide, ks, vgt_spec.')
shading interp
colormap(hot)
xlabel('Time in second', 'FontSize', 12)
ylabel('Frequency(\omega)', 'FontSize', 12)
title('spectrogram created using Gaussian filter', 'FontSize', 12)
xticks([0:1:L])
yticks([-25000:5000:25000])

% Mexican hat
vmt_spec = [];
width = 0.25;
tslide = 0:0.1:L;
for j = 1 : length(tslide)
    mexhat = 2 / (sqrt(3 * width) * pi^(1/4)) * (1 - ((t - tslide(j)) / width).^2)...
    .* exp(-((t - tslide(j)).^2) / (2 * width^2));
    vm = mexhat.* v;
    vmt = fft(vm);
    vmt_spec = [vmt_spec; abs(fftshift(vmt))];
end
subplot(3, 1, 2)
pcolor(tslide, ks, vmt_spec.')
shading interp
colormap(hot)
xlabel('Time in second', 'FontSize', 12)
```

```
ylabel('Frequency(\omega)', 'FontSize', 12)
title('spectrogram created using Mexican hat filter', '
    FontSize', 12)
xticks([0:1:L])
yticks([-25000:5000:25000])

% Shannon
vst_spec = [];
width = 0.4;
for j = 1 : length(tslide)
    shannon = (abs(t - tslide(j)) < width);
    vs = shannon.* v;
    vst = fft(vs);
    vst_spec = [vst_spec; abs(fftshift(vst))];
end
subplot(3, 1, 3)
pcolor(tslide, ks, vst_spec.')
shading interp
colormap(hot)
xlabel('Time in second', 'FontSize', 12)
ylabel('Frequency(\omega)', 'FontSize', 12)
title('spectrogram created using Shannon filter', '
    FontSize', 12)
xticks([0:1:L])
yticks([-25000:5000:25000])


% plot the window with original signal at the center
tau = 4.5;
gabor = exp(-25 * (t - tau).^2);
mexhat = 2 / (sqrt(3 * 0.25) * pi^(1/4)) * (1 - ((t - tau
    ) / 0.25).^2)...
    .* exp(-((t - tau).^2) / (2 * 0.25^2));
shannon = (abs(t - tau) < 0.4);
figure(5)
subplot(3,1,1)
plot(t, v, 'k')
hold on
plot(t, gabor, 'Linewidth', 3)
xlabel('Time in second', 'FontSize', 12)
ylabel('Frequency(\omega)', 'FontSize', 12)
title('centered Gaussian window with width of 25', '
    FontSize', 12)

subplot(3,1,2)
```

```
plot(t, v, 'k')
hold on
plot(t, mexhat, 'Linewidth', 3)
xlabel('Time in second', 'FontSize', 12)
ylabel('Frequency(\omega)', 'FontSize', 12)
title('centered Mexican hat window with width of 0.25', '
    FontSize', 12)

subplot(3,1,3)
plot(t, v, 'k')
hold on
plot(t, shannon, 'Linewidth', 3)
xlabel('Time in second', 'FontSize', 12)
ylabel('Frequency(\omega)', 'FontSize', 12)
title('centered Shannon window with width of 0.4', '
    FontSize', 12)



%%
% Part 2

% Piano
[y,Fs] = audioread('music1.wav');
tr_piano=length(y)/Fs; % record time in seconds
%{
plot((1:length(y))/Fs,y);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Mary had a little lamb (piano)');
%}

L = 16;
v = y';
n = length(v);
t = (1 : length(v)) / Fs;
k= (1/L)*[0:(n/2-1), -n/2:-1];
ks = fftshift(k);
tslide = 0 : 0.1 : L;
vgt_spec = [];
frequency = [];

width = 30;
for j = 1 : length(tslide)
    gabor = exp(-width * (t - tslide(j)).^2);
    vg = v.* gabor;
    vgt = fft(vg);
```

```
    [vmax, index] = max(abs(vgt));
    frequency = [frequency; abs(k(index))];
    vgt_spec = [vgt_spec; abs(fftshift(vgt))];
end
figure(6)
plot(tslide, frequency)
xlabel('time in second', 'FontSize', 15)
ylabel('frequency in hertz', 'FontSize', 15)
title('the center frequencies of the piano', 'FontSize',
    15)


figure(7)
pcolor(tslide, ks, vgt_spec.')
shading interp
colormap(hot)
xlabel('time in second', 'FontSize', 15)
ylabel('frequency(\omega)', 'FontSize', 15)
title('the spectrogram of the piano', 'FontSize', 15)
ylim([0, 1000])
xticks([0:1:L])

%%
% recorder
[y,Fs] = audioread('music2.wav');
tr_rec=length(y)/Fs; % record time in seconds
%{
plot((1:length(y))/Fs,y);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Mary had a little lamb (recorder)');
%}

L = 14.2;
v = y';
v = v(1 : length(v));
n = length(v);
t = (1 : length(v)) / Fs;
k= (1/L)*[0:(n/2-1), -n/2:-1];
ks = fftshift(k);
tslide = 0 : 0.1 : L;
vgt_spec = [];
frequency = [];

width = 30;
for j = 1 : length(tslide)
    gabor = exp(-width * (t - tslide(j)).^2);
```

```
256         vg = v.* gabor;
257         vgt = fft(vg);
258         [vmax, index] = max(abs(vgt));
259         frequency = [frequency; abs(k(index))];
260         vgt_spec = [vgt_spec; abs(fftshift(vgt))];
261     end
262     figure(8)
263     plot(tslide, frequency)
264     xlabel('time in second', 'FontSize', 15)
265     ylabel('frequency in hertz', 'FontSize', 15)
266     title('the center frequencies of the recorder', 'FontSize
        ', 15)
267
268
269     figure(9)
270     pcolor(tslide, ks, vgt_spec.')
271     shading interp
272     colormap(hot)
273     xlabel('time in second', 'FontSize', 15)
274     ylabel('frequency(\omega)', 'FontSize', 15)
275     title('the spectrogram of the recorder', 'FontSize', 15)
276     ylim([0, 3000])
277     xticks([0:1:L])
```