Gábor Transform - Analysis of Audio Files

Quinn Luo

February 10^{th} 2019

Abstract

This report explains the procedure of plotting spectrograms of audio file. I will implement Gábor transform and compare those created by different filters with different width and different time windows. Then I will repeat this process to recognize the music scores of two other clips.

Introduction

A spectrogram is an image that displays frequency content happening over time. Given an audio clip, we should be able to perform time-frequency analysis to extract useful information. In this report, I employ spectrograms to show the nodes and the length and order of nodes playing in given audio files. In the end, I am able to recognize the exact nodes of these clips by associating the frequency in the music scale table.

Background

- **Spectrogram: a trade-off between time and frequency**

  An audio clip stores content as waves over time but we cannot access the frequency

content without performing Fourier transformation. However, by transforming the waves into frequency domain, we lose track of time. That is, Fourier transformation enables us to see all frequencies happening during the clip but we do not know when these frequencies take place and for how long they last.

However, spectrograms manage to store both time and frequency content. By transforming waves to frequency domain only within a time window, we obtain the frequency content during this window, even though we still do not know how these frequencies line up within this time window. And we repeat this process for the next time window, until the end of the track.

Spectrorgram stores content in both time and frequency domain, but neither of them is as accurate as if the wave is stored only in time or frequency domain. That is, we have to lose some information. High resolution in time domain leads to low accuracy in frequency, and vice versa.

- **Gábor transforms**

  To plot the spectrogram, we can perform a step-wise frequency analysis by implementing Gábor transform. Gábor transform is a short-time Fourier transform. By filtering waves within a small time window, performing Fourier transform on that window, and moving to filter the next window, until it reaches the end of the clip, Gábor transform obtains both time and frequency content and stores the information in a two-dimensional matrix.

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

  The transform involves two parameters: translation τ and dilation ω. Translation refers to the location of filter center. We can move the filter along the time axis by

changing value of translation. Dilation is a measure of the width of the filter. The time window gets wider the larger dilation is. Translation and dilation characterizes the Gábor transform, but different there are many different shapes and types of filters to use. In this report, Gaussian filter, Mexican hat filter, and Shannon filter are employed to achieve this short-time window. These filters, though in different shapes, take value of approximately zero outside the time window.

Algorithms and results

Part I: Spectrograms of Handel excerpt

First, I load the wav file as a vector and trim the last element to make the length even. Then, I create $n+1$ linearly spread out points in the duration of this audio clip. These points are linearly scaled into the domain of 2π, and the first n points are shifted because the fast Fourier transform algorithm will swap the left and right halves of these points. The last mode is equivalent to the first one because of the periodic behavior of this 2π domain. There are in total Fs points, number of data points depending on given file.

Then I construct filters around the starting point and slide through to the end by increasing the value of *tslide* from 0 to 9. Different filters are available. For example, Gaussian filter e^{-t^2}, Mexican hat filter $\frac{1}{\sqrt{2\sigma}\pi^{1/4}}(1-(\frac{t}{\sigma})^2)e^{-\frac{t^2}{2\sigma^2}}$, and Shannon filter, a unit step function. Figure 1. below are different filters used in this report.

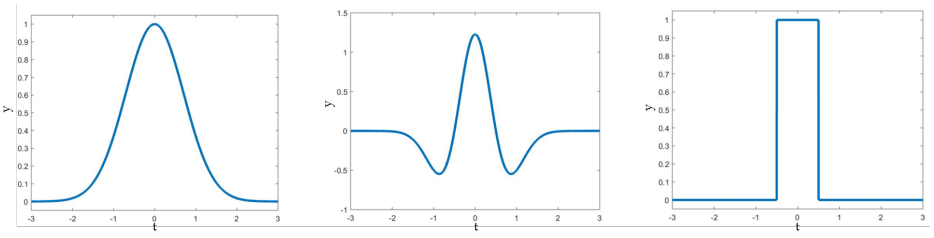

Figure 1: Different filters: Gaussian(left); Mexican hat(middle); Shannon(right)

- **Changing the width of Gaussian filter**

  By setting `strength` to 50, and sliding the Gaussian filter, I transform the waves in every time window into frequency domain and record them in a matrix whose rows are the frequency content in that time slot. I repeat this process with `strength` equal to 1000 and 50000. Figure 2. shows the spectrograms created by procedures above.

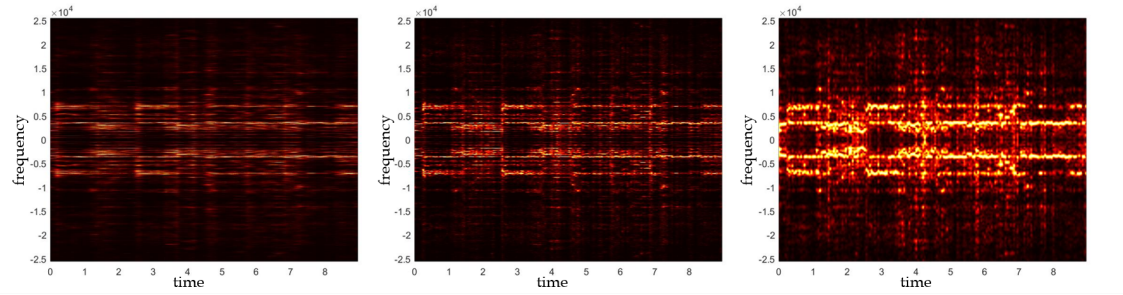

Figure 2: Spectrograms created from Gaussian filter with strength 50(left); 1000(middle); 50000(right)

  The red and orange dots show the location of frequencies. As you can tell, the left spectrogram has all frequencies are clearly separated in the y direction, i.e., frequencies are perfectly distinguishable. However, they are fuzzy in the x direction. The spectrogram in the middle with `strength` 1000 is a great balance between frequency and time. It is clear in terms of both x and y directions. As for the one on the right, hot dots are separated in x direction while fuzzy overlapping in y direction.

- **Oversampling and undersampling**

  There are possibility of oversampling and undersampling, the former using small translation and sampling too much data while the the latter using large translation and sampling not enough data. Figure 3. are examples of oversampling and undersampling. Oversampling makes the spectrogram noisy and hard to read, and undersampling has insufficient information at many time slots.

- **Mexican hat filter and Shannon filter**

  These filters produce the similar results: with right parameters, they can plot spectro-

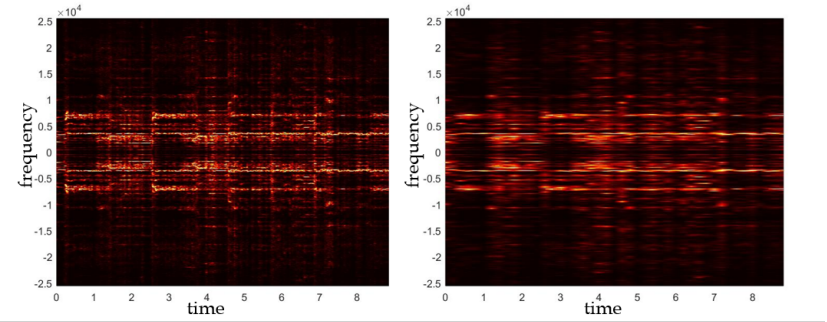

Figure 3: Oversampling(left); Undersampling(right)

grams with good precision in time and frequency.

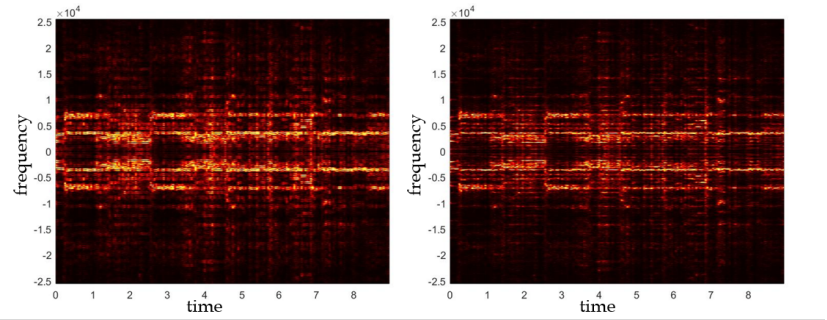

Figure 4: Spectrograms created from Mexican hat filter(left); Shannon filter(right)

Part II: Recognizing nodes

For music1.wav and music2.wav, I repeat the procedure in Part I without trimming the vector because the length is already even): creating Fourier modes, choosing parameters to get a clear spectrogram, and obtaining the matrix which contains the frequency content at each time slot. However, the spectrogram based on this matrix is hardly readable with other frequencies laying above the main one. This is caused by overtone, a phenomenon related to the timbre of the instrument.

Hence, in order to extract out the dominant frequency, I access the location where the maximum frequency takes place and find its frequency in Fourier domain. To note that this is the frequency in the domain of 2π so I must scale it back by dividing it by 2π.

Now, I plot the dominant frequencies at each time slot for both files, obtaining Figure 5 and 6. And I match them with the closest music scale frequency.

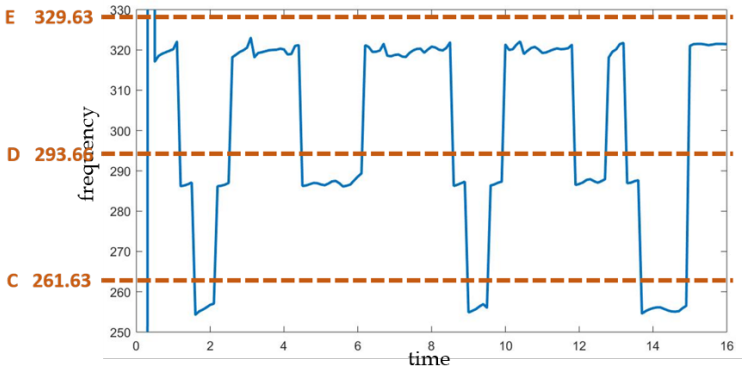

Figure 5: Computed nodes from music1.wav

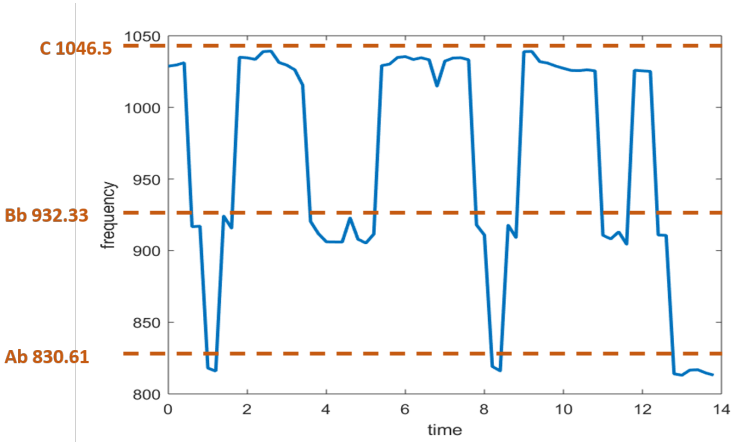

Figure 6: Computed nodes from music2.wav

Conclusion

In conclusion, with audio files given, I create spectrograms by performing short-time Fourier transform, called Gábor transform with appropriate parameters: width and type of filter and width of time window. Moreover, I can find the dominant frequencies at each time slot to recognize the music scores by comparing it to the music scale table.

Appendix I: MATLAB commands explained

- `fft` returns the Fourier transforms of each column of input matrix, working with different dimensions
- `fftshift` swaps the left half and right half of the vector
- `[M, I] = max(A)` stores the maximum value of A as M an its index as I
- `linspace(x1, x2, n)` generates n linearly spaced points between x1 and x2
- `heaviside(a)` creates a function of value 0 if x $<$ t and 1 if x $>$ t
- `pcolor(X, Y, C)` makes a pseudocolor map of C on the grid of X and Y
- `audioread(FILE)` reads an audio file to a vector

Appendix II: MATLAB code

Part I

```
clear all; close all; clc


load handel
v = y'/2;
v = v(1:end- 1);
L = 9; n = length(v);
t2 = linspace(0, L, n+1); t = t2(1:n);
k = (2*pi/L)*[0:n/2-1 -n/2:-1]; ks = fftshift(k);


%Filters
%Gabor/Shannon: translation width; Ricker: sigma
strength = 1000; sigma = 0.5; width = 0.025;
```

```
tslide = 0:0.02:t(end);
vgt_spec = zeros(length(tslide), 73112);
for i = 1:length(tslide)
   %Gaussian filter
   g = exp(-strength*(t - tslide(i)).^2);
   %Mexican hat wavelet
   %g = 2./(sqrt(3.*sigma).*(pi^(1/4))).*(1-((t-tslide(i))./sigma).^2)
   %.*exp(-(t-tslide(i)).^2./(2*sigma.^2));
   %Shannon filter
   %g = heaviside(t-tslide(i)+width).*(1-heaviside(t-tslide(i)-width));
   vt = fftshift(fft(v));
   vg = g.*v;
   vgt = fft(vg);
   vgt_spec(i, :) = abs(fftshift(vgt));

   subplot(3, 1, 1), plot(t, v, 'r', t, g, 'k'); axis([0 9 -1 1])
   xlabel('t'); ylabel('wave')
   subplot(3, 1, 2), plot(t, vg, 'k'); axis([0 9 -1 1])
   xlabel('t'); ylabel('wave')
   subplot(3, 1, 3), plot(t, abs(fftshift(vgt))/max(abs(vgt))); axis([0 9 0 1])
   xlabel('t'); ylabel('frequency')
   drawnow
end

figure(2)
pcolor(tslide, ks, vgt_spec.'./max(abs(vgt_spec.'))), shading interp
colormap(hot)
```

Part II

```
clear all; close all; clc


tr_piano=16; %tr_rec = 14
y=audioread('music1.wav'); %y=audioread('music2.wav');
Fs=length(y)/tr_piano;


S = y(1:end)';
L = tr_piano; n = length(S);
t2 = linspace(0, L, n); t = t2(1:n);
k = (2*pi/L)*[0:n/2-1 -n/2:-1]; ks = fftshift(k);


%Gabor filter
strength = 50;
tslide = 0:0.1:t(end);
S_gt_max = zeros(161, 1);
for i = 1:length(tslide)
   %Gaussian filter
   g = exp(-strength*(t - tslide(i)).^2);
   S_g = g.*S;
   S_gt = fft(S_g);
   [M, I] = max(abs(S_gt));
   S_gt_max(i) = k(I);


   subplot(3, 1, 1), plot(t, S, 'r', t, g, 'k');
   xlabel('t'); ylabel('wave')
   subplot(3, 1, 2), plot(t, S_g, 'k');
```

```
    xlabel('t'); ylabel('wave')
    subplot(3, 1, 3), plot(t, abs(fftshift(S_gt))/max(abs(S_gt)));
    xlabel('t'); ylabel('frequency')
    drawnow
end
figure(2)
plot(tslide, S_gt_max./(2*pi), 'LineWidth', 2)
xlabel('time'); ylabel('frequency')
```

References

Kutz, J. N., (2013). Data-driven modeling scientific computation: Methods for complex systems big data. Oxford: Oxford University Press.