Spectrograms of Music Samples

Aidan Sleavin

When audio is recorded it is converted into a file that is meant to be listened to not visualized. This file looks like a string of numbers and means nothing to a human if looked at. To make the audio file more comprehendible to the human eye it must be displayed in a picture form known as a spectrogram. Exploring with spectrograms lets humans make inferences on the audio file and sometimes possibly know the tune by looking at it.

Overview

A spectrogram looks like a heatmap where along the x axis is time and the y axis are the frequency signature at that time. These plots are generated filtering the audio file to look at a small area in time and determine the frequency at that point in time using a Fourier transform. Then for the entire audio file that process is stepped through many points in time to get a complete picture of the audio signal. This method of taking a series of Fourier transforms filtered on different location is called the Gabor Transform.

There are two important factors when in creating the spectrogram firstly the width of the filter being used, and the number of steps in time that are looked at. The first to understand is the width of the filter, width referring to distance in time. Wider the filter taken the more data the spectrogram shows about the frequency, but the less it shows about where that frequency is in time. At an extreme case the filter being effectively the width of the entire audio file, the spectrogram will have the same frequency signature at each point in time giving little relevant information. As the filter gets smaller the time accuracy gets better but some information is lost about frequency. For example if a audio file is known to have a lot of information at lower frequencies and the filter is very small, when the FFT is taken little to no data will have that low frequency information.

The next important factor being the number of steps. Again looking at the extreme cases can shed some light. If the number of steps is to little, also meaning the distance between time point is high, much information can be lost. If the width of the filter is effectively smaller than the distance between steps all the information contained between steps will be completely gone because it is filtered out every point of the way. If the number of steps is very high the largest effect that will be had is a taxed computer. Implementing the algorithms can take a while to run and if too many steps are looked at without diminishing returns to precision it will take even longer. In both cases it can take trial and error to determine what is the best width of the filter and amount of steps. Depict this figure 2 shows a range

of width of filters in decreasing width, and figure 3 shows a range of steps in increasing amount. Both these show an eight second clip of Handle's messiah.

Implementation and Development

The first step of analyzing an audio file is to load it, in the case of Handel's Messiah being preprogrammed into MATLAB it can be loaded in. Next on line 9 of the Handel code the vector t is made to match up with each recorded point y, n is the number of points in y. on line 10 a vector k is created which is the points that are plotted in frequency space. Because y has an odd number of elements the go from 0:(n-1)/2 -(n-1)/2:-1 with the modifier of 1/tr_handel to make it match up with the unit hertz. "ks" is then the fft of k to use for plotting purposes.

Next a few constants are set up. "a" to be the modifier for the width of the filter, the larger "a" the thinner the filter. A vector "tslide" to be the set of points where the filter is applied to the audio file to create the spectrogram.

The next step is a for loop that at its end creates a matrix "Vgt_spec" which holds all the data for the spectrogram. This is done on lines 19-24 in the Handel code and 33-38 on the Mary had a little lamb code. This is done by stepping through each point of "tslide" and creating a gaussian "g" centered on that time. Instead of a gaussian a decaying sine wave was also tested as a filter, with "g" equal to $Sin(-a * (t - tslide(j)) * \frac{1}{t-tslide(j)})$. The filter is then applied to the audio file and the fft of the file is created to be "Vgt". "Vgt" is the placed into the "Vgt_spec" matrix in the column that relates to that time step. The process is repeated for the length of "tslide". In the pcolor function the input is the "tslide" for the x axis, "ks" for frequency space, and the "Vgt_spec" for the values at all the points. The output is the spectrogram. An important note is to implement the shading interp code for suffer a black plot.

This process is repeated in the Mary had a little lamb code for the two audio files music1.wav and music2.wav where these files need to be got in through audioread. Once the audio file is loaded the creation of the spectrogram is the same as in Handel.

Results

The results from the Handel code in figure 1 and 2 are as predicted. (note the range of the frequency for figure 1 and figure 2 are not the full range of the music but narrowed down to better see stretching). In figure 1 when the width of the filter is too wide the frequency information is stretched

Figure-1 a=1, 10, 100, 1000 times of 200

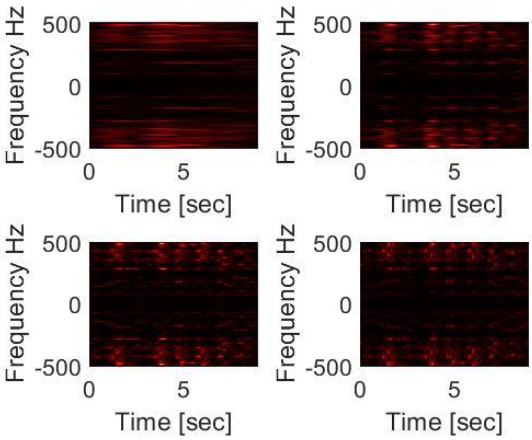
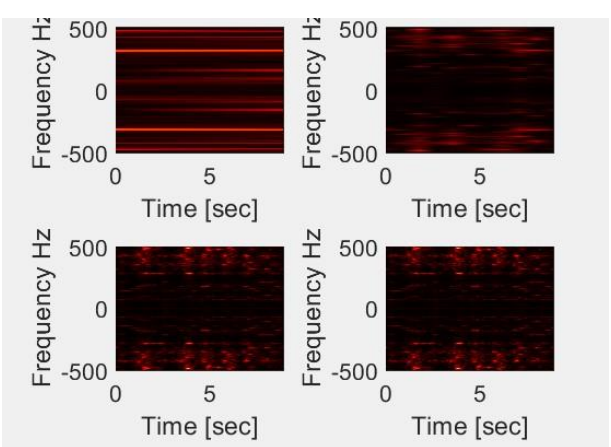


Figure -2 times=1, 10, 100, 300 a of 100



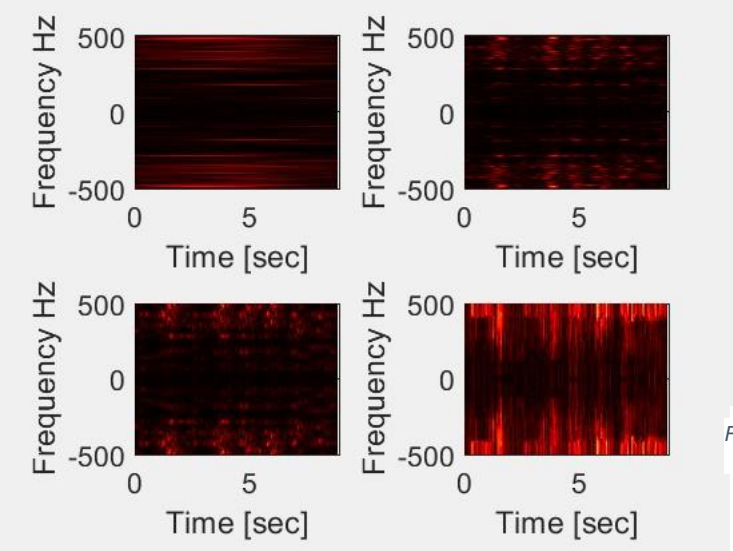


Figure-3: Decaying Sine wave of a=1, 10, 100, 1000

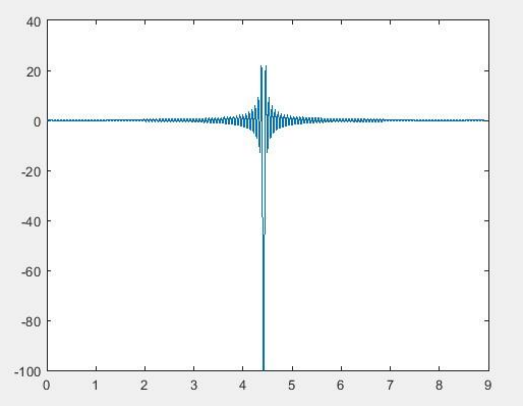


Figure 4: Sine wave of a=100

along the time axis giving bad time accuracy. As the width is narrowed the time resolution is increased but at the extreme case when "a" is 1000 the frequency information begins to be lost as the data is stretched along the frequency axis. The best resolution to a human eye seems to be around "a" equal to 100.

In figure 2 it can be seen at the extreme where only one time point is looked at most of the data is lost due to only a sliver of the audio being viewed. The spectrogram seems to come more into view as "times" is increases. There seems to be no point at which having too many time points hurts the spectrogram. But a max of 300 times is all that could be feasibly tested as the machine running the code had problems displaying the spectrogram and not crashing if "times" got to high.

As for testing a different shape of a filter a decaying sine wave shown in figure 4 (at an arbitrary location) also created a decent result with predictable boundary cases for "a". As with the gaussian as "a" nears zero most time data is lost, and when "a" is large most frequency data is lost. This can be seen in figure 3.

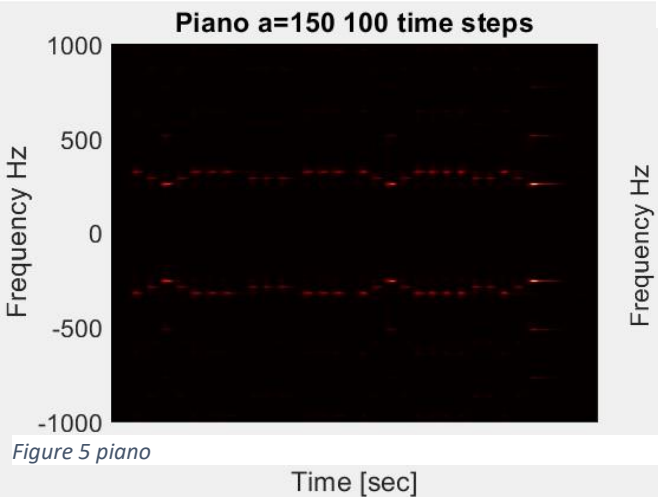


Figure 5 piano

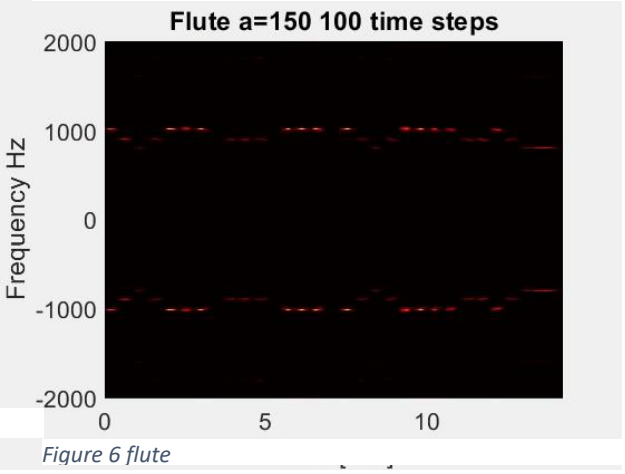


Figure 6 flute

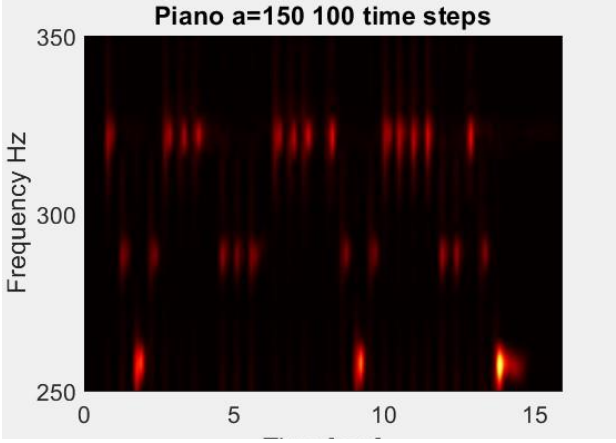


Figure-7 zoomed in on relevant notes

Spectrograms taken for Mary had a little lamb on piano and flute can be seen in figure 5 and 6. These look much clearer than Messiah, which can be attributed to messiah which has a whole orchestra creating a complex sound. Mary had a little lamb was played on one instrument each, and is a relatively simple tune. This simple tune can even be seen as distinct frequencies as the different points in time. the credibility of these two spectrograms is increased by matching up with each other in terms of the pattern they create. The main difference is the flue has frequencies as each point three times higher than that of the piano. This has to do with the flute being a much higher pitched instrument so the same note takes place a few octaves (or frequency steps) higher on the music scale.

Though it is faint to see both the flute and the piano produce frequencies at each time that are multiples of those clearly seen called overtones, this has to do with imperfections in the musical instruments. For example if a piano is playing a note at 400 Hz slight overtones will be recorded at 800, 1200, 1600 and so on, becoming decreasing in magnitude.

The spectrogram can be used to recreate the music notes by comparing to a chart. Using figure 7 which takes the piano and zooms in on the main frequencies 3 distinct notes can be seen. Comparing the frequencies of each of the three notes to a chart, the top row represents E, the middle D, and the bottom C, the sequence goes as [EDCDEEEDDDEEEEDCDEEEEDDEDC]. To show the accuracy of these results I pulled out a PVC flue I made and used an app to figure out where on the flute E, D, and C are and played it back. That can be found here,
https://www.youtube.com/watch?v=IxLz6ZUezNA&feature=youtu.be

In the video it can be seen I am looking behind the camera where I am viewing the spectrogram to know which notes to play. In disclosure I have no musical skill, I cannot read music, and I do not have the ability to hear a piece and know which notes are being played.

Conclusion

Spectrograms can easily be implemented to view the notes in a song and recreate without needing to listen to an audio file. Through use of the Gabor Transform spectrograms can be created for a vector derived from an audio file. The spectrogram can be used to visualize and analyze the music with a possible intent to recreate. As I have shown the spectrogram can be used to visualize and replace sheet music for those with no musical skill.

Appendix A.

- [y,Fs]=audioread: this will read in an audio file and turn it into a vector y and return Fs the frequency the audio is captured at. Such that the time of the audio file is equal to length(y)/Fs
- fft: this takes the Fourier transform of a vector returning values in frequency space
- pcolor: this is a plotting function that let data be visualized on a 2d plane with a colored intensity.
- Shading inter: sets the type of shading to do for the plot. It interpolates the color for all pixels based on known value.
- colormap(): changes what spectrum of colors is used. In this "hot" is used for the entire paper.

Appendix B.

Handel:

```
1  clear; close all; clc;
2
3  load handel
4  y= y(:);
5  v = y;
6  tr_handel=length(y)/Fs;
7
8  n=length(y);
9  t=(tr_handel/n:tr_handel/n:tr_handel); t=t(:);
10 k=(1/tr_handel)*[0:(n-1)/2 -(n-1)/2:-1]; ks=fftshift(k);
11 %%
12 A = [100];
13 for i=1:length(A)
14 a=A(i);
15 times=200;
16 tslide=0:tr_handel/times:tr_handel;
17 Vgt_spec = zeros(length(tslide),n);
18
19 for j=1:length(tslide)
20     g=exp(-a*(t-tslide(j)).^2);
```

```
21      Vg=g.*v;
22      Vgt=fft(Vg);
23      Vgt_spec(j,:) = fftshift(abs(Vgt)); % We don't want to scale it
24  end
25
26
27  subplot(1,1,i)
28  pcolor(tslide,ks,Vgt_spec.')
29  shading interp
30  set(gca,'Ylim',[-500 500],'Fontsize',16)
31  xlabel("Time [sec]"); ylabel("Frequency Hz");
32  colormap(hot)
```

Mary had a little lamb:

```
1  clear; close all; clc;
 2
 3  [y,Fs] = audioread("music1.wav");
 4  tr_piano=length(y)/Fs; % record time in seconds
 5  %plot((1:length(y))/Fs,y);
 6  %xlabel("Time [sec]"); ylabel("Amplitude");
 7  %title("Mary had a little lamb (piano)");
 8  %p8 = audioplayer(y,Fs); playblocking(p8);
 9
10
11  n=length(y);
12  t=linspace(0,tr_piano,n+1); t=t(1:n);t=t(:);
13  k=(1/tr_piano)*[0:n/2-1 -n/2:-1]; ks=fftshift(k);
14  %%
15  close; clear all; clc;
16  [y,Fs] = audioread("music2.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)");
%p8 = audioplayer(y,Fs); playblocking(p8);


n=length(y);
t=linspace(0,tr_piano,n+1); t=t(1:n);t=t(:);
k=(1/tr_piano)*[0:n/2-1 -n/2:-1]; ks=fftshift(k);
%%
a = 150;
times=100;
tslide=0:tr_piano/times:tr_piano;
Vgt_spec = zeros(length(tslide),n);
%%
for j=1:length(tslide)
    g=exp(-a*(t-tslide(j)).^2);
    Vg=g.*y;
    Vgt=fft(Vg);
    Vgt_spec(j,:) = fftshift(abs(Vgt)); % We don't want to scale it
end
%%
pcolor(tslide,ks,Vgt_spec.')
shading interp
set(gca,'Ylim',[-2000 2000],'Fontsize',16)
colormap(hot)
xlabel("Time [sec]"); ylabel("Frequency Hz");
title("Flute a=150 100 time steps");
```