AMATH 482 Homework 2

Maxime Didascalou

February 10, 2021

Abstract

This project uses the Gabor transform and filtering in order to find out the score for the guitar in the introduction of "Sweet Child O' Mine" by the Guns N' Roses, and for both the guitar and bass in "Comfortably Numb" by Pink Floyd.

1 Introduction and Overview

The process of finding the frequencies of different sounds overlapping each other and evolving in time has several limitations, and this project tries to overcome them to try and get the scores for the instruments in the guitar in the introduction of "Sweet Child O' Mine" by the Guns N' Roses (easier since it only has one instrument), and for both the guitar and bass in the instrumental section of "Comfortably Numb" by Pink Floyd.
The report begins with some theoretical background for the processes that I used, and also explains those limitations and why they occur. Then there is an explanation of the algorithms I implemented and the results I got.

2 Theoretical Background

As we learned from our textbook [1], one can take the Gabor transform of a function $f(t)$:

$$\tilde{f}_g(\tau, k) = \int_{-\infty}^{\infty} f(t)g(t-\tau)e^{-ikt}dt. \tag{1}$$

And its inverse takes a function $\tilde{f}_g(\tau, k)$ and converts it to a function of time:

$$f(t) = \frac{1}{2\pi||g||_2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \tilde{f}_g(\tau, k)g(t-\tau)e^{ikt}dkd\tau. \tag{2}$$

Taking a Gabor transform is essentially the same as taking a Fourier transform (which rewrites a function in terms of it's frequency components), but filtered around τ. The function $g(t)$ determines the filter and a common choice is to use a Gaussian, but anything can be chosen as long as g is real and symmetric and its L_2-norm is set to unity.
The Gabor transform is useful in signal analysis because it allows us, with some degree of accuracy, to pinpoint at what time certain frequencies are present in the signal. This is done by simply choosing different values for τ that cover the length of our signal, filtering around each point using our choice of g, and then taking the Fourier transform to find out what frequencies are present around that point.
One caveat of this method is that when the filter is wide, you can get a lot of frequency information out of the transform but the localization in time won't be accurate, since the filter covers a bigger range. And on the other hand, if the filter is very narrow, the localization in time will be accurate, but the frequency information won't since the shape of the filter itself will be more significant compared to the underlying frequencies.

We can also clean up a function (get rid of noise) or focus on a certain frequency by filtering in frequency space. For example, you can multiply the Fourier transform of a function by a Gaussian:

$$F(k) = e^{-\tau(k-k_0)^2}, \tag{3}$$

where τ regulates the width of the filter and k_0 is the central frequency that you want to keep. This essentially focuses on that central frequency and damps everything else. You can then use the inverse Fourier transform on the filtered function to get a cleaner result.
In practice, we are often given discrete points of a signal and not continuous functions, so we have to use the discrete Fourier Transform (DFT): Given a sequence of N equally spaced points $\{t_0, t_1, \ldots, t_{N-1}\}$, the DFT is defined as:

$$\hat{t}_k = \frac{1}{N} \sum_{n=0}^{N-1} t_n e^{\frac{2\pi i k n}{N}}. \tag{4}$$

This however is restricted to: $k = -N/2, -N/2+1, \ldots, -1, 0, 1, \ldots, N/2-1$.
The built in MATLAB function `fft` uses the fast Fourier Transform (FFT) which essentially cuts in half the vector of data points which gives 2 smaller DFTs, and then the process is repeated untill the vectors can't be cut in half anymore. This reduces the complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$.

3 Algorithm Implementation and Development

This section will cover all the code that was used for this project:

1. Setting up all the data and variables.
2. Get bigger picture for both songs using Gabor transform
3. Combining Gabor transform and frequency filtering
4. Isolating the bass in "Comfortably Numb" (whole clip)
5. Combining Gabor transform and filtering overtones. This needs to be done since when instruments play a note they often send out several frequencies. Say f is the fundamental frequency (the main one), then the overtones are integer multiples of f.

Algorithm 1: Import and set up variables

Import data for both songs
y = vector of data points for each song
`n` = number of points for each song (length of y)
find `tr` = length of each song in seconds (n/sampling frequency)
`t` = `n` equally spaced points up to `tr`
Set up frequencies as described in Theoretical background
Scale them by 1/`tr` to convert to Hz

Algorithm 2: Bigger Picture

`a` = 1
τ = set of equally spaced points between 0 and `tr`
for each τ **do**
  `g` = Gaussian filter centered at τ (width controlled by `a`)
  `yFiltered` = g × y
  Take Fourier Transform of `yFiltered` using MATLAB function `fft`
  Shift the returned values to the conventional format (`fft` returns the values for the negative frequencies last)
end for
Create pseudo-color plot of the transforms with the frequencies on the vertical and τs on the horizontal axes
Try different values for `a` and τ to clean up the result
Set up the boundaries of the plot for desired frequencies

Algorithm 3: Finding Bass Score

`a` = 1
τ = set of equally spaced points between 0 and `tr`
`aFreq` = 0.001
τ_{Freq} = 100 (the bass is played at around 100 Hz, which we know from Algorithm 2)
for each τ **do**
  `g` = Gaussian filter centered at τ (width controlled by `a`)
  `gFreq` = Gaussian filter centered at τ_{Freq} (width controlled by `aFreq`)
  `yFiltered` = g × y
  Take Fourier Transform of `yFiltered` using MATLAB function `fft`
  Shift the returned values to the conventional format
  Multiply by `gFreq`
end for
Create pseudo-color plot of the transforms with the frequencies on the vertical and τs on the horizontal axes
Try different values for `a`,`aFreq` , τ, and τ_{Freq} to clean up the result
Set up the boundaries of the plot for desired frequencies

Algorithm 4: Isolate Bass

Take the `fft` of y
Shift it to conventional values
Multiply it by a filter such that it is 0 for frequencies not in [-130 -70] or [70 130] (these intervals come from Algorithm 2)
Shift the values back to what `ifft` understands
Take the `ifft`

Algorithm 5: Finding guitar score (Pink Floyd)

`a` = 1
τ = set of equally spaced points between 0 and `tr`
`aOvertone` = 0.05
for each τ **do**
  `g` = Gaussian filter centered at τ (width controlled by `a`)
  `yFiltered` = g × y
  Take Fourier Transform of `yFiltered` using MATLAB function `fft`
  Shift the returned values to the conventional format
  Find the frequency (`fundamental`) of the Guitar that is playing the same notes as the bass (one octave higher)
  Create filters for the overtones of that frequency: 1 - Gaussian centered at `fundamental` with width controlled by `aOvertone`
  Multiply shifted transform with those filters
end for
Create pseudo-color plot of the results with the frequencies on the vertical and τs on the horizontal axes
Try different values for `a`,`aFreq` , τ, and τ_{Freq} to clean up the result
Set up the boundaries of the plot for desired frequencies

4 Computational Results

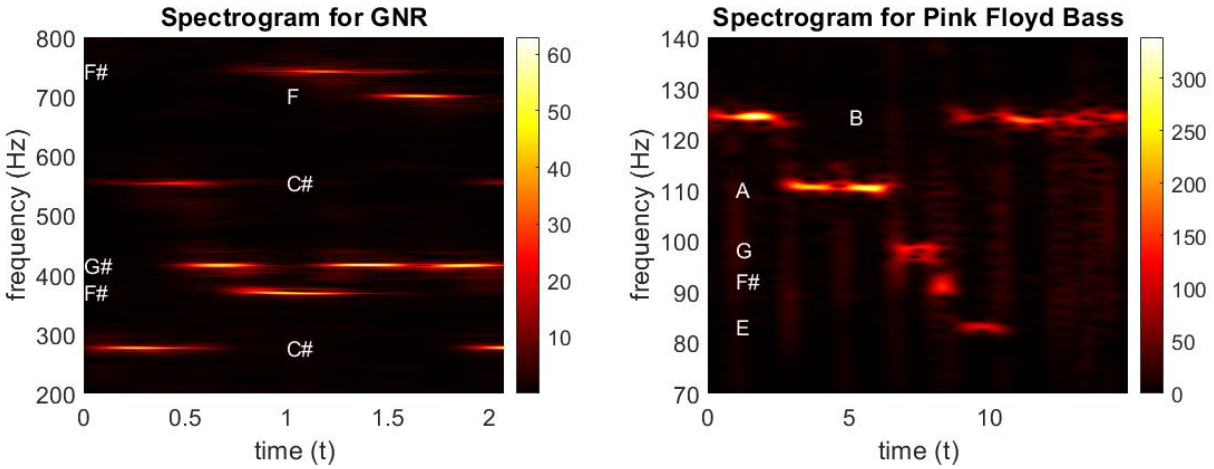
4.1 Sweet Child O' Mine

There are four bars in this clip and the guitar follows the same theme in all of them (repeated twice). The notes for the first bar are C# C# G#F #F# G# F G#, but since guitars are often tuned a half step down, the guitarist was probably playing D D A G G A F# A. The only changes for the rest of the clip are that the first note in the second bar is D# (or E if we account for the tuning) and the first note in the third bar is F# (or G if we account for the tuning).
I used Figure 1a to get the notes for the first bar. I just needed Algorithm 2 to get that result since the clip only contains guitar and doesn't need to be filtered.

4.2 Comfortably numb

The bass pattern repeats four times in this clip. The main theme is |B B |A A |G F# E |B B |. There are minor things that change such as how many times each note is played in a row. The spectrogram is in Figure 1b.

The guitar score was much more challenging to find as it was mixed with lots of overtones from the instruments of lower frequency such as the bass. Another guitar was also playing the same melody as the bass, just one octave higher, so to get the guitar solo score I had to filter out the overtones of that second guitar. The resulting Spectrogram is not a full score but it does show the notes that were the most accentuated: F#, then E, then D, and then finally B (this is over the first quarter of the clip). As you can see on Figure 2, the D was getting filtered out since it landed on one of the overtones of the underlying guitar. However, since it is much stronger than anything else, I think it is safe to assume that it is also part of the solo.



(a) First half of first bar of GNR

(b) Bass pattern for Pink Floyd

Figure 1: Spectograms for bass (Pink Floyd) and guitar (GNR)

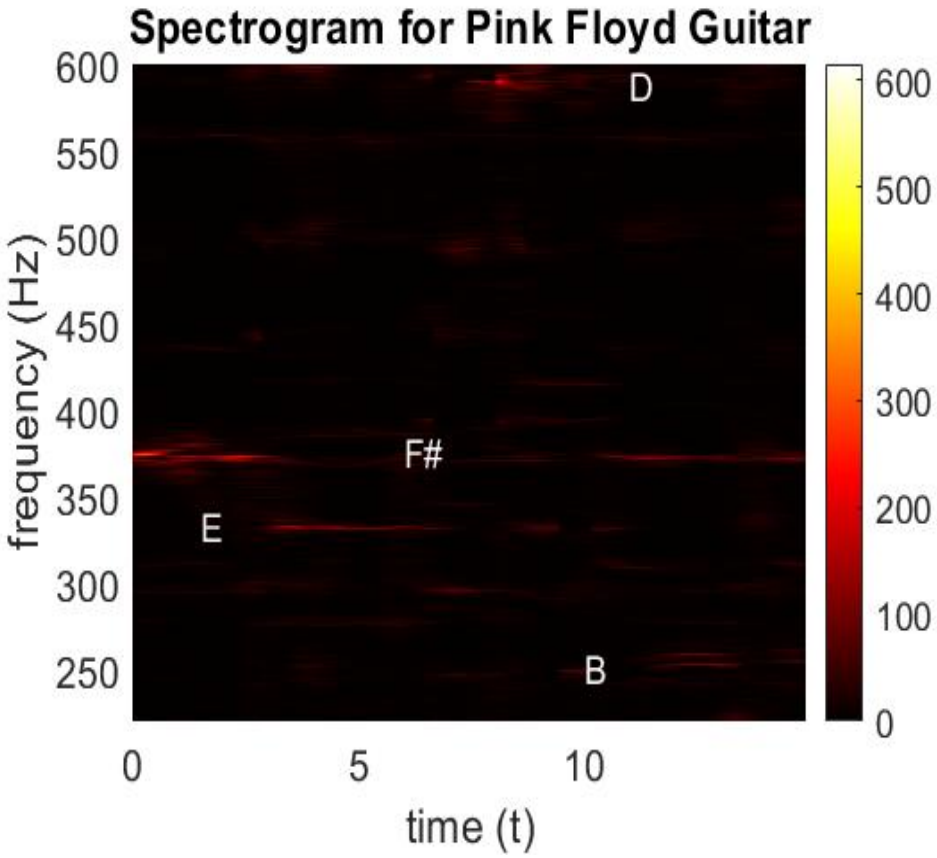


Figure 2: Guitar for Pink Floyd

5 Summary and Conclusions

After taking the Fourier transforms of those songs filtered at different times (Gabor transforms) I was able to figure out what notes were being played and when. It was also useful to filter out unwanted frequencies in frequency space such as lower/higher frequencies or overtones of different instruments. In conclusion, the Gabor transform and filtering are great tools for this purpose, but there are some accuracy limitations, as explained in the theoretical background, and thus sometimes the rhythm or the notes are not easily identifiable. In this case, the part that was the most challenging was the guitar in "Comfortably Numb" since it was mixed in with a lot of overtones and some of the notes were played very fast. But the main notes were still distinguishable.

References

[1] Jose Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

Appendix A MATLAB Functions

- `y = linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- `[y,Fs] = audioread(filename)` reads data from the file named `filename`, and returns sampled data, `y`, and a sample rate for that data, `Fs`.
- `Y = fft(X)` computes the discrete Fourier transform (DFT) of `X` using a fast Fourier transform (FFT) algorithm.
- `X = ifft(Y)` computes the inverse discrete Fourier transform of `Y` using a fast Fourier transform algorithm.
- `Y = fftshift(X)` rearranges a Fourier transform `X` by shifting the zero-frequency component to the center of the array.
- `X = ifftshift` undoes the result of `fftshift`.
- `[M,I] = max(A,[],_,'linear')` returns the linear index into `A` that corresponds to the maximum value in `A`.
- `pcolor(X,Y,C)` creates a pseudocolor plot (used for spectrogram) using the values in matrix `C`. `X` and `Y` specify the x and y coordinates of the vertices. They must match the size of `C`.

Appendix B MATLAB Code

```
figure(1)
[y_guns, Fs_guns] = audioread('GNR.m4a');
y_guns = y_guns(1:floor(length(y_guns)/4))';
%y_guns = y_guns';
n = length(y_guns);
tr_guns = n/Fs_guns; % record time in seconds
plot((1:n)/Fs_guns,y_guns);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Sweet Child O Mine');

t2 = linspace(0,tr_guns,n+1);
t = t2(1:n);
k = (1/tr_guns)*[0:n/2-1 -n/2:-1];
ks = fftshift(k);
% p8 = audioplayer(y_guns,Fs_guns); playblocking(p8);

figure(2)
[y_floyd, Fs_floyd] = audioread('Floyd.m4a');
%y_floyd = y_floyd';
y_floyd = y_floyd(1:floor(length(y_floyd)/4))';
n2 = length(y_floyd);
tr_floyd = n2/Fs_floyd; % record time in seconds
plot((1:n2)/Fs_floyd,y_floyd);
xlabel('Time [sec]'); ylabel('Amplitude');
title('Comfortably Numb');

t2_f = linspace(0,tr_floyd,n2+1);
t_f = t2_f(1:n2);
ks_f = (1/tr_floyd)*[-n2/2:n2/2-1];
k_f = ifftshift(ks_f);
% p8_f = audioplayer(y_floyd,Fs_floyd); playblocking(p8_f);
```

Listing 1: Setup

```
a = 40;
tau = 0:0.03:tr_guns;

for j = 1:length(tau)
   g = exp(-a*(t - tau(j)).^2); % Window function
   Sg = g.*y_guns;
   Sgt = fft(Sg);
   Sgt_spec(:,j) = fftshift(abs(Sgt)); % We don't want to scale it
end

figure(3)
pcolor(tau(1:70),ks,Sgt_spec(:,1:70))
shading interp
set(gca,'ylim',[200 800],'Fontsize',16)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')
```

Listing 2: Big Picture

```
a = 5;
tau = 0:0.1:tr_floyd;
a_fbig = 0.001;
tau_fbig = 100;
a_overtone = 0.05;

for j = 1:length(tau)
   g = exp(-a*(t_f - tau(j)).^2); % Window function
   % g_f = exp(-a_fbig*(ks_f-tau_fbig).^2); % For algorithm 3

   Sg = g.*y_floyd;
   Sgt = fft(Sg);
   Sgtshift = fftshift(abs(Sgt));
   [m, ind] = max(Sgtshift(330406:331464));
   freq = ks_f(ind + 330406)*2;

   filter1 = 1 - exp(-a_overtone*(ks_f-freq*1).^2);
   filter2 = 1 - exp(-a_overtone*(ks_f-freq*2).^2);
   filter3 = 1 - exp(-a_overtone*(ks_f-freq*3).^2);
   filter4 = 1 - exp(-a_overtone*(ks_f-freq*4).^2);
   filter5 = 1 - exp(-a_overtone*(ks_f-freq*5).^2);

   Sgt_spec(:,j) = Sgtshift.*filter1.*filter2.*filter3.*filter4.*filter5;
end

figure(4)
pcolor(tau,ks_f,Sgt_spec)
shading interp
set(gca,'ylim',[220 600],'Fontsize',16)
colormap(hot)
colorbar
xlabel('time (t)'), ylabel('frequency (Hz)')
```

Listing 3: Use for Algorithms 3 and 5

```
freqs = fftshift(fft(y_floyd));
z = zeros([1 length(ks_f)]);
for i = 1:length(ks_f)
    if (-70 > ks_f(i) && ks_f(i) > -130) || (70 < ks_f(i) && ks_f(i) < 130)
        z(i) = freqs(i);
    end
end
z2 = ifft(ifftshift(z));
p8_filtered = audioplayer(abs(z2),Fs_floyd); playblocking(p8_filtered);
```

Listing 4: Isolating bass sound