AMATH 582 Homework 2: Classic Rock and Roll Shredding

Brady Griffith

Abstract

Spectrograms made suing the Gabor transform provide insight into how a signal's spectrum changes with time. Musical scores are just instructions of which frequencies to play for a certain amount of time, so should be identifiable from the spectrogram. In this work, I do just that with clips from *Sweet Child O' Mine* by Guns N' Roses and *Comfortably Numb* by Pink Floyd. It was possible to identify the correct score for the guitar solo in the Guns N' Roses song. In the Pink Floyd song the bass riff could be fully identified, along with parts of the guitar solo.

Introduction and Overview

In 1977, Ian Dury introduced the immortal phrase "Sex and drugs and rock and roll" declaring them to be "very good indeed". Only the third one is suitable for time series analysis, so rock and roll will be the subject of this work. The objective is to determine the score from clips from of songs *Sweet Child O' Mine* by Guns N' Roses and *Comfortably Numb* by Pink Floyd.

The ability to make spectrograms is already built onto most audio editing software, so this project really is just a demonstration of the technique. This method does have the advantage of being able to choose the colormap used.

Theoretical Background

The fundamental tool for this analysis is the discrete Gabor transform. This gives a changing spectrum as over time by taking the fourier transform of a window around that a series of time steps. In the continuous form, the Gabor transform of a signal $f(t)$ is

$$\tilde{f}(t,\omega) = \int_{-\infty}^{\infty} f(\tau)\bar{g}(\tau - t)e^{-i\omega\tau}\,\mathrm{d}\tau \tag{1}$$

where $g(\tau - t)$ is the windowing function.

The choise of windowing function can have effects on the legibility of the spectrogram. One of the biggest considerations of a window function is the tradeoff between resolution in time, σ_t, and frequency, σ_f. The uncertainty principle conditions that

$$\sigma_t^2\sigma_f^2 \geq \frac{1}{16\pi^2}. \tag{2}$$

The minimum uncertainty is achieved by gaussian windows, which is why those will be used throughout the project.

I also make use of frequency filtering to isolate the bass. Because sharp filters introduce ringing and other artifacts some care is taken with filter design. The bass is lower pitch than the other instruments, so a low pass filter is used. I make it fourth order (−80dB/decade). The filter function is

$$H(f) = \left(\frac{1}{f/f_0 + 1}\right)^4 \tag{3}$$

where f_0 is the cutoff frequency.

Algorithm Implementation and Development

The first step is to import the songs in a way that gives a numpy array. I use the scikit-sound package, which gives an object with both the sound data as a numpy array, and the sample rate.

The Gabor transform is preformed using the `stft` function in the scipy signal package. I'd like to control slightly different parameters than the arguments of this function, so some processing is needed. The windows parameter takes care of the gaussian window with the appropriate width. There is also nperseg which is the size of the segment around each sample time to preform the transform over. It must be large enough that the gaussian has nearly gone to 0 by the edges. I use the nearest (in log space) power to two samples to 5 times the gaussian width. The time sampling frequency is handled by noverlap, which is how much each window overlaps with the previous. I set this to be the segment width minus one quarter the gaussian width. This produces spectra every quarter gaussian width.

There are 3 different spectrograms produced which each have a unique choice of window width. To examine the guitar riff in *Sweet Child O' Mine*, the riff is at around 4 notes per seconds and requires a time resolution > 0.125s. The lowest note is $C4\sharp$ at 277Hz which requires a frequency resolution of > 8Hz to distinguish from the next note. This is more than easily satisfied by the uncertainty principle, so I split the difference in time/frequency and use 0.05s window widths, which gives a 6.4Hz frequency resolution.

This tradeoff is somewhat more complicated for *Comfortably Numb*. First I consider the bass riff which is within the range of 50Hz to 110Hz Hz. In order to distinguish notes at the low end, a resolution > 1.4Hz. However, the bass riff is mush slower than the others at around 60 BPM. For these reasons, a longer 0.1s window is used.

The guitar riff in that song has a some rapid variations, so a higher time resolution is needed. This requirement comes at the expense of higher frequency resolution, but as mentioned in the the first riff, it's still possible to reduce that and distinguish notes. A .04s window is used.

The bass rift exists below 140Hz, so that is used for the low pass cutoff. The entire song is fourier transformed with the `rfft` function in numpy. The filter is

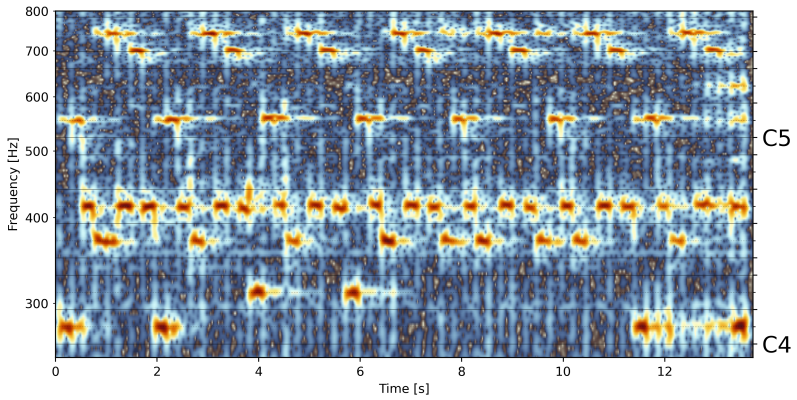


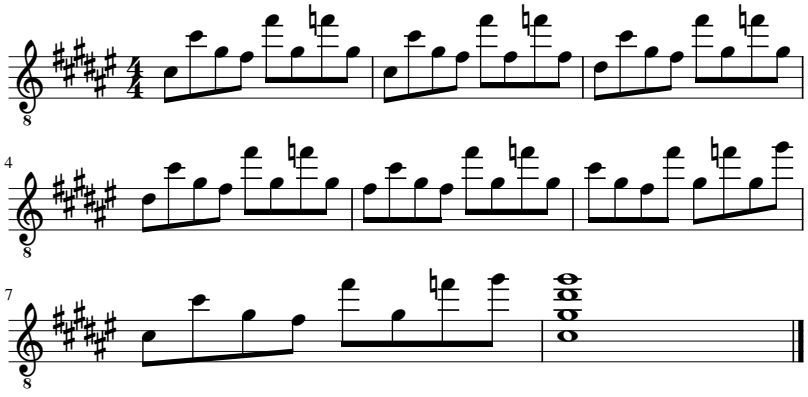
Figure 1: The spectrogram of the intro to *Sweet Child O' Mine* by Guns N' Roses. This range highlights the guitar riff.

multiplied against this and the song is moved back into the time domain with irfft . The array is converted into a sound object and exported as a wav file.

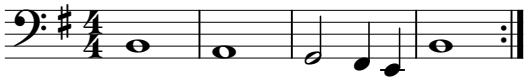
Computational Results

A spectrogram focusing on each of the three instruments is now presented. I make use of a SciVisColor wave colormap in order to highlight structure across the full range of intensities.

The first spectrogram is for *Sweet Child O' Mine* and is in figure 1. It is very easy to identify the notes of the riff. As I transcribed it, the score is:



A spectrogram for the bass in *Comfortably Numb* is presented in figure 2. This part of the audio is highlighted with a low pass filter, and can be listened to here. As I transcribed it, the score is:



The guitar riff comes out less desirably. The spectrogram is presented in figure 3. I'm not able to write out a score as clearly as I did for the last two, but I can highlight a series of notes that should appear in the score in a certain order, basically a the score but missing some notes. This sequence is: $F4\sharp$, $E4$, $D5$, $B3$, $D5$, $F5\sharp$, $D5$, $B4$, $D5$, $E4$, $G4$, $D5$, $F5\sharp$, $A4$, $B3$, $F4\sharp$, $B4$, $E4$, $D5$.

Summary and Conclusions

Using the Gabor transform, the spectrograms of the two clips were produced. For the guitar in *Sweet Child O' Mine* and the bass in *Comfortably Numb*, the notes are clearly identifiable. I have nearly an exact match between my score and the notes of published tabs [1] [2]. The guitar solo in the Pink Floyd song was not clear enough to identify the score. That part of the spectrum suffered from severe pollution by the bass overtones. In addition, the solo as given by published tabs is much more complicated than the other two. What could be identified was an accurate subset of the song.

The bass was isolated with a low pass filter, which was very successful. The output file is almost entirely the sound of the bass. This was useful to confirming that the bass score was correct.

References

[1] "Tab by guns n' roses." https://tabs.ultimate-guitar.com/tab/guns-n-roses/sweet-child-o-mine-tabs-12657.

[2] "Bass tablature for comfortably numb by pink floyd.." https://www.bigbasstabs.com/pink_floyd_bass_tabs/comfortably_numb_bass_tab.html.

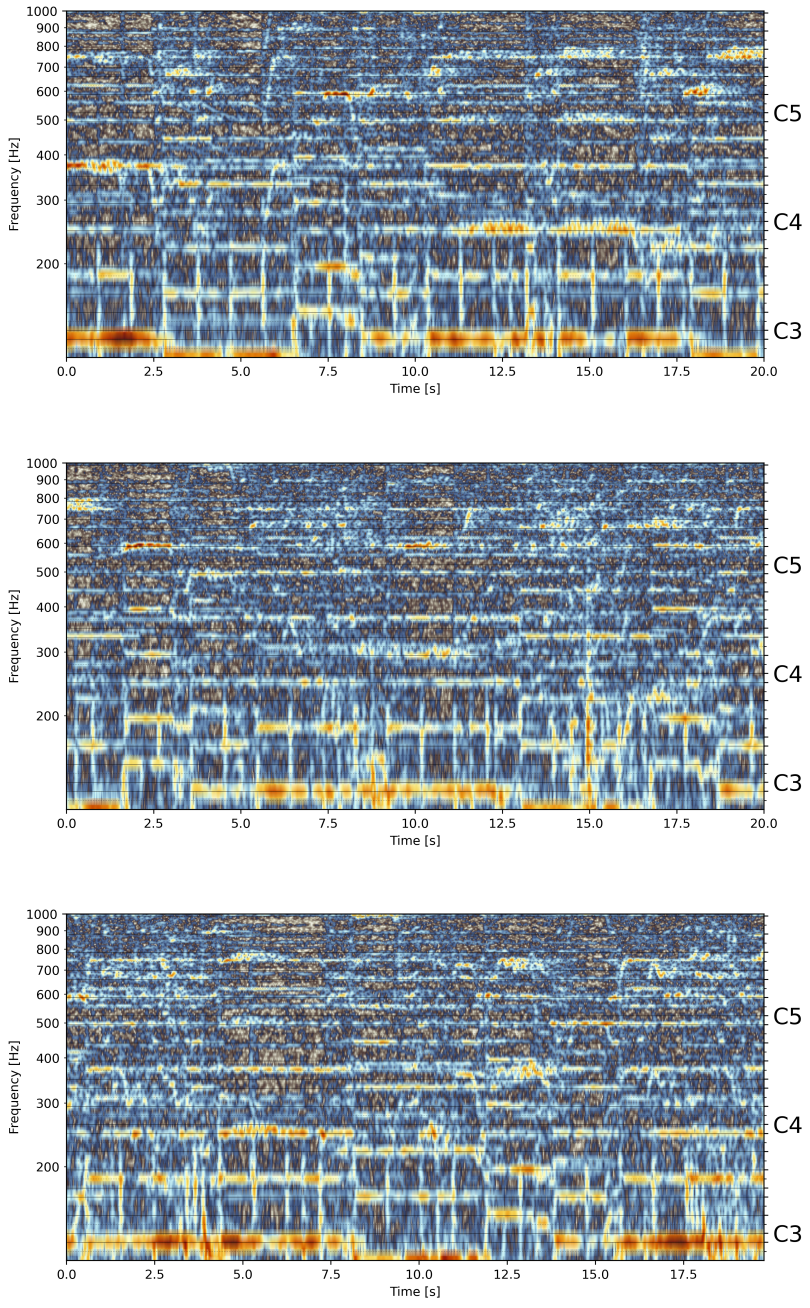


Figure 2: The spectrogram of the second guitar solo in *Comfortably Numb* by Pink Floyd. This range highlights the bass riff.

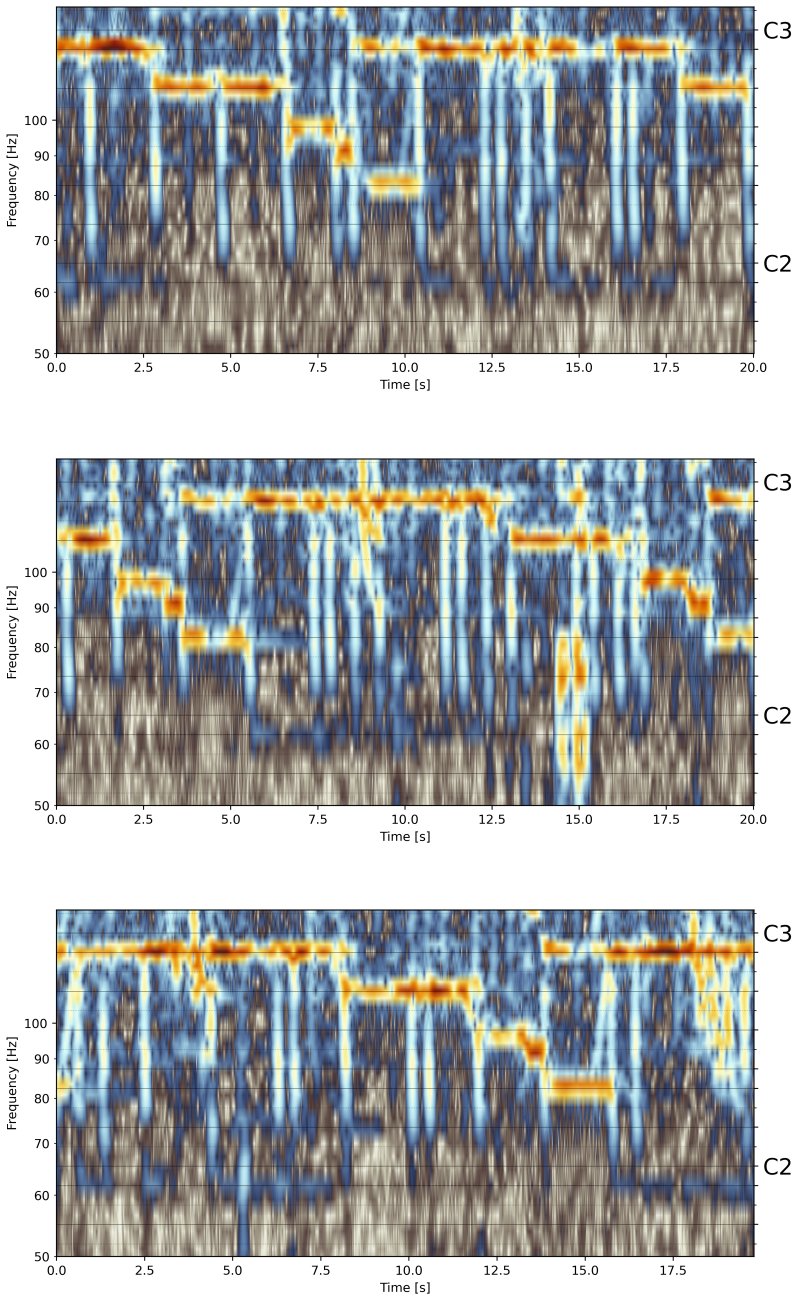


Figure 3: The spectrogram of the second guitar solo in *Comfortably Numb* by Pink Floyd. This range highlights the guitar riff.

This code is hosted on github here.

Python Functions

spectrum

plot_spectrum

```
plot_spectrum(f, t, S, name, freq_range=(50, 2000))
```

Takes a provided rectangular array of spectrum vs time and creates a meshplot with the markings for notes of the equal tempered scale.

Arguments:

- `f` *array_like* - A 1D array of frequencies for the spectrum in P
- `t` *array_like* - Time samples of the spectrum.
- `S` *array_like* - An array of spectra at different times. Should be of shape (len(t), len(f)).
- `name` *str* - The output file same. The plot will be saved in "HW2/figures/name.png".
- `freq_range` *tuple* - The ends of the frequency range to plot. Defaults to (50 Hz, 2000 Hz).

get_spectrum

```
get_spectrum(data, rate, width)
```

Produces the Gabor transform of the data provided using a gaussian window with the width provided. The transform is given in timesteps of 1/4 the width.

Arguments:

- `data` *array_like* - The time series to be transformed.
- `rate` *float* - The sampling rate (#/s) used for data.
- `width` *float* - The width of the gaussian window in seconds.

Returns:

- `f` *ndarray* - Array of sample frequencies.
- `t` *ndarray* - Array of sample times.
- `S` *ndarray* - The Gabor transform of data.

low_pass

```
low_pass(data, rate, cutoff)
```

Applies a 4th order low pass filter to a time series.

Arguments:

- `data` *array_like* - The time series to be filtered.

- `rate` *float* - The sample rate of data.
- `cutoff` *float* - Cutoff frequency for the filter in Hz.

run_gnr

`run_gnr()`

Generates the spectrum of the Sweet Child O' Mine clip.

run_floyd_guitar

`run_floyd_guitar()`

Generates the spectrum of the guitar in the Comfortably Numb clip.

run_floyd_bass

`run_floyd_bass()`

Generates the spectrum of the bass in the Comfortably Numb clip, and produces a filtered clip of just the bass.

Python Code

spectrum.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import ticker
from scipy import signal
from sksound import sounds
import cm_xml_to_matplotlib as cm


def plot_spectrum(f, t, S, name, freq_range=(50, 2000)):
    """Takes a provided rectangular array of spectrum vs time and creates a
    meshplot with the markings for notes of the equal tempered scale.

    Args:
        f (array_like): A 1D array of frequencies for the spectrum in P
        t (array_like): Time samples of the spectrum.
        S (array_like): An array of spectra at different times. Should be of
            shape (len(t), len(f)).
        name (str): The output file same. The plot will be saved in
            "HW2/figures/name.png".
        freq_range (tuple): The ends of the frequency range to plot. Defaults
            to (50 Hz, 2000 Hz).
```

```
"""
if np.shape(S) != (len(f), len(t)):
    raise ValueError('S provided is of incorrect shape')

# Load in the note names/frequencies
notes = pd.read_csv('HW2/data/notes.csv')
in_range = (notes['hertz'] > freq_range[0]) & \
           (notes['hertz'] < freq_range[1])
notes = notes[in_range]
natural = notes['note_flat'] == notes['note_sharp']
notes['note_flat'] = notes['note_flat'] + notes['octave'].astype(str)

# Start Plotting
fig, ax_freq = plt.subplots(figsize=(10, 5))
ax_note = ax_freq.twinx()  # Axis to mark notes on

for ax in [ax_freq, ax_note]:
    ax.set_ylim(*freq_range)
    ax.set_yscale('log')

# Don't mark with scientific notation
ax_freq.get_yaxis().set_major_formatter(ticker.ScalarFormatter())
ax_freq.get_yaxis().set_minor_formatter(ticker.ScalarFormatter())

# Label only C in each octave, but mark all natural notes
ax_note.set_yticks(notes['hertz'][natural], minor=False)
labels = [n if n[0] == 'C' else '' for n in notes['note_flat'][natural]]
ax_note.set_yticklabels(labels, fontsize=18)

# Only mark sharps/flats as minor ticks without labeling
ax_note.set_yticks(notes['hertz'][~natural], minor=True)
ax_note.get_yaxis().set_minor_formatter(ticker.NullFormatter())

# Add lines across to mark the notes
ax_note.grid(which='major', ls='-', c='k', lw=.5, alpha=.5)
ax_note.grid(which='minor', ls=':', c='k', lw=.5, alpha=.4)

ax_freq.set_xlabel('Time [s]')
ax_freq.set_ylabel('Frequency [Hz]')

# Proportional to PSD in db, but magnitude isn't displayed
psd = np.log(np.abs(S)**2)

vmin = np.percentile(psd, 50)

# wavecmap = cm.make_cmap('HW2/code/12-4w_heir1_ccc1.xml')
```

```
    wavecmap = cm.make_cmap('HW2/code/3-3w_grblrd_ccc1.xml')
    ax_freq.pcolormesh(t, f, psd, cmap=wavecmap, shading='gouraud', vmin=vmin)
    fig.savefig(f'HW2/figures/{name}.png', dpi=400)


def get_spectrum(data, rate, width):
    """Produces the Gabor transform of the data provided using a gaussian
    window with the width provided. The transform is given in timesteps of
    1/4 the width.

    Args:
        data (array_like): The time series to be transformed.
        rate (float): The sampling rate (#/s) used for data.
        width (float): The width of the gaussian window in seconds.

    Returns:
        f (ndarray): Array of sample frequencies.
        t (ndarray): Array of sample times.
        S (ndarray): The Gabor transform of data.

    """
    window_width_pts = rate*width
    # Choose the size of the segment to be the nearest (in log space) power of
    # two to 4x the width of the filter.
    segment_width = pow(2, round(np.log2(5*window_width_pts)))
    overlap = segment_width - window_width_pts//4

    return signal.stft(data,
                       rate,
                       ('gaussian', window_width_pts),
                       nperseg=segment_width,
                       noverlap=overlap)


def low_pass(data, rate, cutoff):
    """Applies a 4th order low pass filter to a time series.

    Args:
        data (array_like): The time series to be filtered.
        rate (float): The sample rate of data.
        cutoff (float): Cutoff frequency for the filter in Hz.
    """
    S = np.fft.rfft(data)
    f = np.fft.rfftfreq(len(data), 1/rate)
    H = 1/(f/cutoff + 1)**4
    return np.fft.irfft(H*S)
```

```
def run_gnr():
    """Generates the spectrum of the Sweet Child O' Mine clip."""
    audio = sounds.Sound('HW2/data/GNR.m4a')
    f, t, S = get_spectrum(audio.data, audio.rate, .05)
    plot_spectrum(f, t, S, 'GNR-Spectrum', (250, 800))


def run_floyd_guitar():
    """Generates the spectrum of the guitar in the Comfortably Numb clip."""
    audio = sounds.Sound('HW2/data/Floyd.m4a')
    data = audio.data
    split_pts = [0, 20*audio.rate, 40*audio.rate, len(data)]

    for n, (left, right) in enumerate(zip(split_pts[:-1], split_pts[1:])):
        f, t, S = get_spectrum(data[left:right], audio.rate, .04)
        plot_spectrum(f, t, S, f'Floyd-Guitar-Spectrum-{n}', (110, 1000))


def run_floyd_bass():
    """Generates the spectrum of the bass in the Comfortably Numb clip, and
    produces a filtered clip of just the bass.
    """
    audio = sounds.Sound('HW2/data/Floyd.m4a')
    data = audio.data
    split_pts = [0, 20*audio.rate, 40*audio.rate, len(data)]

    for n, (left, right) in enumerate(zip(split_pts[:-1], split_pts[1:])):
        f, t, S = get_spectrum(data[left:right], audio.rate, .1)
        plot_spectrum(f, t, S, f'Floyd-Bass-Spectrum-{n}', (50, 140))

    bass_audio_data = low_pass(data, audio.rate, 140)
    bass_audio = sounds.Sound(inData=bass_audio_data, inRate=audio.rate)
    bass_audio.write_wav('HW2/output/Floyd-Bass.wav')


if __name__ == '__main__':
    run_gnr()
    run_floyd_bass()
    run_floyd_guitar()
```